ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE COMPUTADORES

DISEÑO Y DESARROLLO DE UN SISTEMA DE GESTIÓN DE INFORMACIÓN TURÍSTICA

DESIGN AND DEVELOPMENT OF A TOURIST INFORMATION MANAGEMENT SYSTEM

Realizado por

Alejandro Aznar Alcaide

Tutorizado por

José Antonio Montenegro Montes

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, septiembre de 2018

Fecha defensa:

El Secretario del Tribunal

Resumen

Las aplicaciones actuales son generalmente sistemas distribuidos que se ejecutan en un entorno cliente, responsable de atender al usuario a través de una interfaz, y un entorno servidor, que almacena los datos e implementa la lógica de negocio. Ambas partes se comunican generalmente implementando una API de datos en el servidor de forma que se puedan realizar consultas y operaciones sobre los datos de forma remota desde cada cliente.

Aunque existen varias tecnologías para resolver esta necesidad, en los últimos años se ha generalizado el uso de REST como sistema estándar de intercambio y manipulación de datos. En muchos casos es una opción válida, pero plantea algunos inconvenientes en aplicaciones que son intensivas en el uso de datos o cuando coexisten diferentes aplicaciones que utilizan el mismo servidor como proveedor de datos. Estos problemas, junto a la necesidad de avanzar más rápido en productos cada vez más complejos, ha motivado la creación de nuevas formas de interactuar con los datos.

Estas circunstancias son el origen de este proyecto, con el que se pretende crear desde cero un sistema rápido y estable ayudado de una serie de herramientas como son NodeJS, GraphQL o React para facilitar el uso de una nueva forma de tratar los datos y mostrárselo al usuario (que éstos no ocupen mucho espacio de memoria para que la transmisión sea menos costosa y más rápida, el navegar por la aplicación sin recargas innecesarias y de forma fluida, etc.).

Palabras claves

Bitbucket, CSS, GitHub, GraphQL, HTML, HTML5, JavaScript, MongoDB, NodeJS, Programación reactiva, React, Robo 3T, Robomongo.

Abstract

Generally, current applications are distributed systems which are executed on the client's side, where users are provided with an interface, hosted in a server environment, where data is saved and business logic is implemented. Both sides are usually communicating with the data's API. Thanks to this API, users can execute data requests and some operations remotely from each client.

Although there are several technologies to solve this need, in the last few years REST has been established as a standard exchange and data manipulation system. In many cases, this is a valid tool. However, this poses certain disadvantages in applications whose data traffic is too heavy, or when different applications use the same server to get data. These problems, considering the need to access the data faster in increasingly complex products, has motivated the creation of new ways to interact with data.

These circumstances are the origin of this project, which tries to set up a rapid and stable system with tools such as NodeJS, GraphQL or React. As a result, we will be able to get and show data from a heavy database to the final user, who will benefit. For example, the user won't be slowed down because of a heavy downloading to find specific information, and he will have a fluent navigation without unnecessary reloads of websites.

Keywords

Bitbucket, CSS, GitHub, GraphQL, HTML, HTML5, JavaScript, MongoDB, NodeJS, React, Reactive programming, Robo 3T, Robomongo.

Agradecimientos

Quisiera dar las gracias a Monte, por tenderme la mano sin dudarlo cuando me embarqué en este proyecto, a pesar de la gran carga de trabajo que tenía.

A mis amigos y compañeros de trabajo, por esos momentos de apoyo y ánimos.

A mi gente de La Línea, mis hermanos y mis padres, que gracias a ellos se ha podido finalizar este viaje tiempo con sus empujoncitos de ánimo, comidas, horas impagables del cuidado de mis hijos... ¡Os quiero!

Pero sobre todo, agradecer en especial a Rocío, mi pilar del día a día. Sin ella esto no habría sido posible. Tu preciado tiempo, tus ánimos, tu paciencia y comprensión, tu generosidad, tu cariño, tu amistad... simplemente tú.

A mis hijos Carlos y Lucía, y a Rocío. Os amo.

Índice

Índice de figuras

1. Introducción

Con la creación de este proyecto pretendemos crear desde cero un sistema rápido y estable, partiendo de una extensa base de datos de donde mostraremos al usuario toda la información deseada con el menor de los costes posibles.

Para lograrlo, nos ayudaremos de herramientas como NodeJS, GraphQL o React. Iremos más allá de las tradicionales bases de datos relacionales, usando las no relaciones como MongoDB, y cambiaremos las ya conocidas llamadas REST por una novedosa API de GraphQL, optimizando tiempos y recursos en cada petición. Todo ello de cara al usuario final con un interfaz creado con React, donde la programación realizada en Javascript será la base fundamental de este proyecto, permitiendo al usuario una navegación fluida e intuitiva.

Para realizar la tarea propuesta ha sido necesaria una gran inversión de tiempo en el aprendizaje de estas nuevas tecnologías, así como la selección de las herramientas adecuadas para ser usadas en el proyecto.

1.1 Objetivos

Los principales objetivos de este proyecto son:

- Migrar una tecnología de desarrollo web obsoleta y con gestión de datos relacionales a una totalmente actual con sistemas de datos no relacionales.
- Estudio y aprendizaje de tecnologías de desarrollo web actuales.
- Estudio y aprendizaje de almacenamiento de datos en sistemas no relacionales, también llamados NoSQL [1] o “no sólo SQL”.
- Demostrar la potencia de JavaScript como lenguaje único de desarrollo.
- Aplicar y valorar los resultados de aplicar el paradigma de la programación reactiva en la programación web de última generación.
- Simular fielmente el sistema de trabajo de un equipo de trabajo usando sistemas de respaldo y control de versiones.

Para conseguir este objetivo, se plantea el desarrollo de una página web con las siguientes características:

- Deben usarse tecnologías no propietarias durante todo el desarrollo del proyecto.
- Debe usar una base de datos no relacional como sistema de almacenamiento de toda la información.
- Debe estar desarrollada íntegramente en JavaScript, tanto en el lado cliente como servidor.
- A modo del tradicional REST (Representational State Transfer), debe usar GraphQL [2] [3] como API (Application Programming Interface) para la petición y devolución de datos en la comunicación cliente-servidor.
- Usar sistemas de respaldos y control de versiones conforme avanza la programación de la aplicación web. De esta forma estaremos cubiertos ante cualquier imprevisto que nos ocurra, y podremos rescatar una versión lo más actual y estable posible, minimizando así una posible penalización de tiempo y perdida de código de programación.
- Ante un gran volumen de información, el usuario debe navegar de forma fluida por entre las distintas páginas que se ofertan.
- Optimización de los recursos ofertados para que el usuario descargue los mínimos datos posibles y éste no se vea penalizado sea cual sea el tipo de su conexión.

A pesar de lo rápido que avanza la tecnología en este campo de la programación web y la gran cantidad de herramientas surgidas, se ha intentado que el desarrollo de este proyecto se ponga en práctica, entre otros, conocimientos de última generación en base de datos, programación y desarrollo web, utilizando en cada uno de estos ámbitos las técnicas y metodologías más actuales posibles.

1.2 Motivación

La razón principal por la que se desarrolla este proyecto viene determinada por los siguientes motivos:

- Deseo de aprender y profundizar en el conocimiento del desarrollo de aplicaciones en una plataforma moderna en constante actualización que tiene un gran presente y futuro, como es el desarrollo de aplicaciones en JavaScript.
- Especialización en el desarrollo de aplicaciones reactivas y sistemas dirigidos por eventos.
- Aplicación de los conceptos fundamentales adquiridos durante la carrera y la vida laboral con unas tecnologías completamente nuevas y sin experiencia previa.
- Puesta en valor de la importancia del desarrollo con herramientas basadas en código abierto.

A nivel personal, la implantación del resultado del proyecto en mi actual lugar de trabajo supone una de las mayores motivaciones. Todo ello conlleva establecer nuevos proyectos de desarrollo web con tecnologías actuales, estar lo más actualizados posible en el mercado laboral de nuestro sector, y poder ir abandonando poco a poco todas las tecnologías obsoletas que seguimos usando, aunque será lento debido a la gran cantidad de proyectos que tenemos y que poseen vida propia, pues requieren de mantenimiento, nuevas mejoras y funcionalidades, y por supuesto respaldo técnico.

1.3 Esquema de la memoria

En esta memoria se expone, en una serie de capítulos, los pasos que han llevado a la consecución de los objetivos marcados:

- En el presente capítulo se hace una introducción, descripción de contenidos, motivación y una presentación del contenido de la memoria.
- En el capítulo 2 se analiza el uso de la tecnología utilizada en el panorama actual de programación web, y las conclusiones que justifican el desarrollo de este proyecto.
- En el capítulo 3 se describe la planificación del proyecto, incluyendo una descripción de la metodología utilizada, fases en que se ha dividido, evaluación del riesgo y el plan de comunicaciones que se ha seguido.

- En el capítulo 4 se ha hecho un estudio y evaluación de las principales herramientas utilizadas en la programación web, así como de los posibles lenguajes de desarrollo, tecnologías y software utilizados finalmente en este proyecto.
- En el capítulo 5 se aborda la fase del desarrollo del software, explicando cómo han sido implementadas cada una de las partes.
- En el capítulo 6 se incluyen los pasos realizados para llevar a cabo la implantación del proyecto.
- En el capítulo 7 se exponen las conclusiones obtenidas del trabajo realizado y las posibles vías de ampliación o mejora del proyecto.
- Por último, se incluye un listado completo de referencias utilizadas en el desarrollo.

2. Situación actual

2.1 Tendencias en los últimos años

Hoy en día, JavaScript está presente en prácticamente cualquier ámbito:

- Sistemas operativos (WinJS)
- Desarrollo móvil (PhoneGap)
- Servidores de internet (NodeJS)
- Bases de datos (MongoDB)
- Administración de sistemas, tanto Linux como Windows, etc.

De hecho, lo más habitual es usarlo como único lenguaje de programación para montar una página web, desde el cliente hasta el servidor, pasando por la base de datos. Es lo que se le conoce ya con el nombre de "stack MEAN" (MongoDB + Express + AngularJS + NodeJS). En esta imagen de CampusMVP [85] puede verse claramente cómo es el flujo del entorno de programación web.

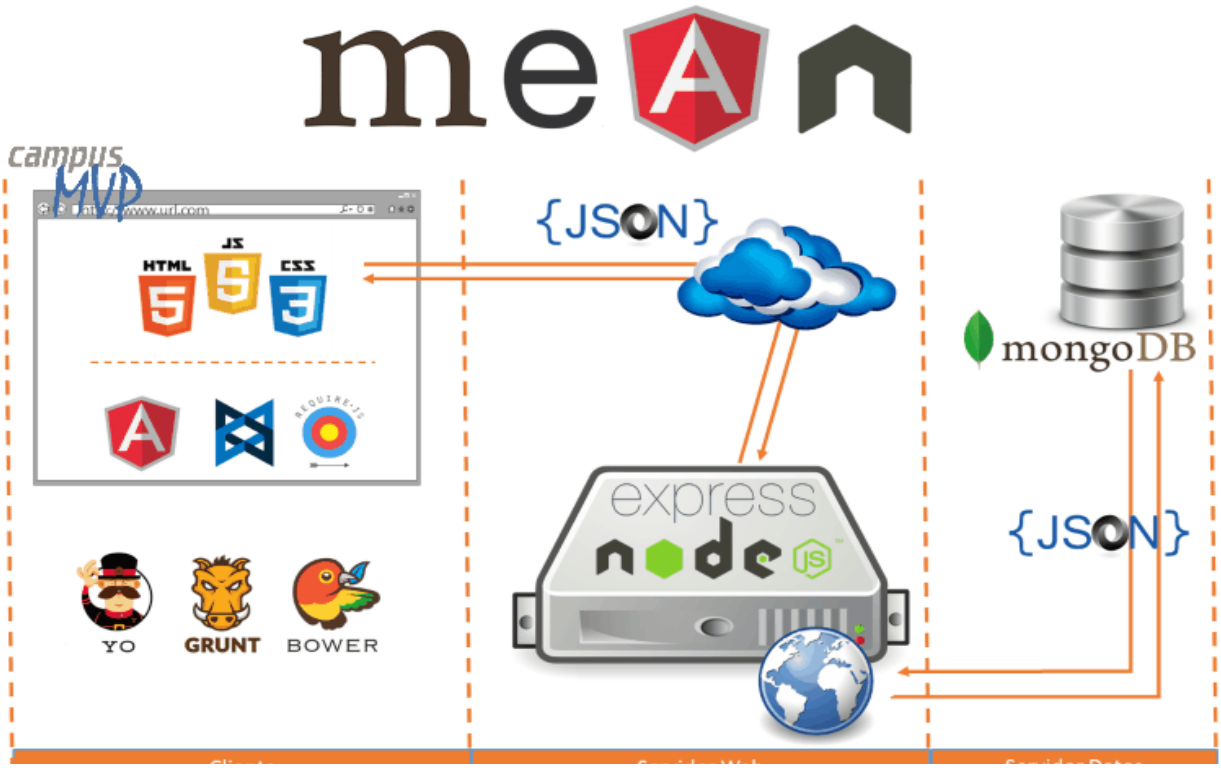


Figura 1 - Flujo del entorno de programación web

Es por eso por lo que todo aquel que forme parte de esta rama tecnológica como es la programación web, debe manejar JavaScript en mayor o menor medida, y cuantos más conocimientos tenga de las nuevas tecnologías para usar este lenguaje de programación, más ventajosa será su posición respecto a otros programadores para conseguir su estatus laboral deseado.

2.2 Herramientas utilizadas

Para la realización de este proyecto, se han utilizado servicios gratuitos y herramientas no propietarias en todas las secciones.

2.2.1 Lado cliente

Se ha utilizado para la visualización de la aplicación web:

- HTML5
- CSS3
- React [4]

2.2.2 Lado servidor

Para la programación del lado servidor se ha utilizado:

- NodeJS
- Express
- GraphQL
- Heroku como servicio online que nos permite subir nuestro código para hacerlo de carácter público y navegable.

2.2.3 Almacenamiento de datos

Para el almacenaje de datos se ha utilizado:

- MongoDB
- Robomongo (Robo 3T)
- mLab como servicio online que permite el subir nuestros datos en la nube.

2.2.4 Programación y ayuda online

Para poder llevar a cabo ha sido indispensable el uso de un editor de textos para programar el código, y una serie de elementos de ayuda, bien sea para adquirir más conocimientos sobres problemas encontrados, o bien para facilitar el desarrollo de la aplicación web:

- Visual Studio Code como editor.
- Bitbucket como sistema de respaldo de control de versiones.
- Librerías varias de origen no propietario descargadas desde GitHub [5]
- Guías, blogs, foros, tutoriales y video tutoriales varios para el aprendizaje y/o ayuda para la solución de problemas encontrados.

2.3 Conclusiones

Tras analizar distintas herramientas a usar en nuestro proyecto, se pueden obtener las siguientes conclusiones:

- No hay un “santo grial” entre todas las herramientas que hay actualmente en este panorama de programación web, y conforme avanza el tiempo, aparecen más y más frameworks o librerías que permiten al programador en mejor o menor medida desarrollar su trabajo. Ya sea AngularJS, React, Vue [6], etc., es el arquitecto del equipo de trabajo quien debe analizar detenidamente si una u otra le es más favorable dependiendo de la envergadura y funcionalidad del proyecto.
- Esta selección de herramientas no ha sido así desde el primer momento. Conforme ha ido creciendo el proyecto y nos hemos ido encontrando problemas o restricciones, se ha procedido al cambio más beneficioso para el desarrollo de éste.
- Hoy en día es indispensable el tener algún tipo de respaldo de control de versiones de nuestro código. En nuestro caso hemos usado Bitbucket frente a GitHub debido a que permite almacenar proyectos privados con la cuenta gratuita que ofrecen.

3. Planificación

La planificación es una parte esencial de cualquier proyecto de diseño o desarrollo. Al esbozar el alcance del proyecto desde el inicio, se identifican y evitan los obstáculos que de otro modo podrían entorpecer el desarrollo de nuestro trabajo.

Para la elección de la metodología de trabajo se ha tenido en cuenta que se trata de un proyecto individual y que los requisitos recogidos inicialmente están sujetos a variaciones durante el desarrollo. Por estos motivos, se ha decidido que el método de desarrollo debía alejarse de los ciclos de vida tradicionales tales como el modelo secuencial, utilizados para proyectos de gran envergadura con requisitos muy bien definidos.

En los siguientes apartados se describe brevemente cuáles son los principales ciclos de vida evaluados y cuál se ha considerado más apropiado para el proyecto, así como la metodología seguida en su desarrollo, el sistema de gestión de tareas y la herramienta con la que se ha implementado toda la gestión.

En último lugar se describe la técnica usada para la gestión del tiempo y mejorar la concentración y productividad en la ejecución de las tareas que componen el proyecto.

3.1 Metodología

Una metodología [7] es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en la implementación de nuevos sistemas de información. Habitualmente divide el trabajo en fases que se pueden puede dividir a su vez en otras, permite elegir las técnicas más apropiadas en cada momento del proyecto y ayuda a planificar, gestionar, controlar y evaluar un proyecto.

3.1.1 Modelo

Dependiendo de la organización y solapamiento entre las fases, se pueden diferenciar los siguientes modelos de ciclo de vida.

3.1.1.1 Modelo predictivo

Tradicionalmente se ha utilizado en el desarrollo los ciclos de vida predictivos (también conocidos como clásicos u orientados a la planificación) en los cuales el alcance, plazo y costo se determinan lo antes posible en el ciclo de vida del proyecto y los esfuerzos se orientan a cumplir con los compromisos establecidos en cada uno de estos factores.

Normalmente estos proyectos se organizan en una serie de fases secuenciales o consecutivas, donde cada una de las fases se enfoca en un subproducto o actividad concreta. El trabajo de una fase es muy diferente al del resto y por lo tanto el equipo de proyecto va variando dependiendo de las fases.

Al inicio la dirección del proyecto se centra en definir el alcance y definir un plan detallado de las actividades necesarias. A partir de aquí el trabajo se orienta para cumplir con la planificación. Cualquier cambio en el alcance del proyecto se debe gestionar de forma explícita y, habitualmente, conlleva la revisión de la planificación y la aceptación formal del nuevo plan.

Este modelo no se ajusta a las circunstancias de este proyecto, ya que sería necesario que el producto final estuviera bien definido y existiera previamente un conocimiento bastante amplio sobre la forma de construirlo.

3.1.1.2 Modelo iterativo e incremental

Los ciclos de vida iterativos son aquellos en los cuales se repiten las actividades del proyecto en iteraciones en las que se desarrollan las fases de la cascada estándar, pero se trabaja únicamente sobre un subconjunto de funcionalidad. La entrega total del proyecto se divide en subsistemas priorizados y como resultado final se aumenta el entendimiento del producto final. Si además en cada iteración se mejora el producto, se considera que el ciclo de vida es incremental.

Se opta por los ciclos de vida iterativos cuando es necesario gestionar objetivos poco definidos o de una alta complejidad o cuando la entrega parcial del producto es clave para el éxito.

Aunque este modelo podría ser utilizado, tampoco se considera el más adecuado ya que no existe en el proyecto una relación detallada de requisitos y se desarrolla con tecnologías en las que no se tiene ninguna experiencia previa.

3.1.1.3 Modelo adaptativo

La diferencia principal del ciclo adaptativo con el iterativo e incremental, es que en este último se repiten las actividades del proyecto en fases, en las que se va aumentando la calidad del producto, mientras que en el adaptativo las actividades se modifican, mueven, eliminan o adelantan en función a las necesidades del momento.

Como en el modelo anterior, es clave contar con una lista ordenada del trabajo pendiente dado que desde allí y siguiendo el orden establecido se irá realizando el trabajo. Dicha lista ordenada se denomina habitualmente backlog. Cada vez que se identifique una nueva tarea se incluye en esta lista en la posición que le corresponda según su prioridad. Así, el backlog es donde se gestionarán los cambios, representados por nuevas tareas, modificaciones de tareas existentes, eliminaciones de tareas o reordenación de éstas en la lista.

Este modelo se considera el más adecuado, ya que permite crear un prototipo base en el que se irán incluyendo características en sucesivas fases. En cada una de ellas no será necesario realizar los pasos habituales un sistema en cascada, pero se verificará su funcionamiento mediante su depuración y prueba. El proceso de considera finalizado cuando la aplicación cumple todos los objetivos definidos.

3.1.2 Metodología ágil

La metodología ágil [8] es un marco metodológico de trabajo que plantea mejorar la eficiencia en la producción y la calidad de los productos, tener la capacidad de respuesta al cambio en los productos y sus definiciones, y ofrecer un mejor resultado a través de la entrega temprana y la retroalimentación continua.

Para seguir el modelo anterior se utiliza esta metodología ya que es la que mejor se adapta a un proyecto con requisitos sujetos a grandes variaciones.

3.1.2.1 Lean Software Development

Para seguir este modelo adaptativo se ha seleccionado esta metodología, que puede resumirse en estos siete principios [17]:

- **Eliminar desperdicios.** Eliminar todo lo que no añade valor, como código y funcionalidades innecesarias, retraso en el proceso de desarrollo de software, requisitos poco claros, burocracia, comunicación interna lenta, documentación innecesaria, etc.
- **Amplificar el aprendizaje.** El desarrollo de software se considera un proceso de aprendizaje continuo, es decir, el mejor enfoque para encarar una mejora en el ambiente de desarrollo es ampliar el aprendizaje.
- **Decidir lo más tarde posible.** En el desarrollo de software los mejores resultados se alcanzan con un enfoque basado en que se pueden retrasar ciertas decisiones tanto como sea posible, hasta que éstas se puedan basar en hechos y no en suposiciones y/o pronósticos inciertos.
- **Entregar lo más rápido posible.** Cuanto antes se entregue el producto final sin defectos considerables, más pronto se pueden recibir comentarios que se incorporan en la siguiente iteración. Por tanto, cuanto más cortas sean las iteraciones y mejor es el aprendizaje.
- **Capacitar y potenciar al equipo.** Aunque este principio no se ha podido aplicar al ser un proyecto individual, este principio intenta que los directivos aprendan a escuchar a los desarrolladores, de manera que éstos puedan explicar mejor qué acciones podrían tomar, así como ofrecer sugerencias para mejorar.
- **Construir con calidad.** En el proceso de desarrollo se debe instaurar la meta de encontrar y corregir los defectos tan pronto como sea posible. Para asegurar la detección de estos defectos, es conveniente poseer un completo conjunto de pruebas a lo largo de todo el ciclo de vida del desarrollo y ejecutarlas automáticamente y de manera periódica.
- **Ver el todo.** Los sistemas de software no son solamente la suma de sus partes, sino también el producto de sus interacciones. Cuanto más grande sea el sistema, más serán las organizaciones que participan en su desarrollo, más partes son las desarrolladas por diferentes equipos y mayor es la importancia de tener bien definidas las relaciones entre los

diferentes proveedores con el fin de producir una buena interacción entre los componentes del sistema.

Las principales ventajas que aporta esta metodología al proyecto son:

- Las entregas pueden ser todo lo frecuente que se desee, llegando al punto de hacerlas cada vez que se termina una tarea. Esto permite hacer validaciones más frecuentes. Además, se evita acumulación de defectos entre cada versión debido a diferentes tareas y sus interferencias. Por otro lado, también es más fácil detectar errores.
- La definición de una tarea se puede postergar hasta el momento en el cual se decide abordarla. Esto favorece la resistencia al cambio, pues no se necesita hacer inversión de esfuerzo con anticipación. Mientras más tiempo hay entre la definición y la ejecución de una tarea, más riesgo existe que dicha definición caduque o que la tarea pierda prioridad.
- Mantener una lista de tareas ordenadas por importancia permite ver fácilmente cómo progresa el desarrollo, y en caso extremo, cancelar alguna tarea que ya no sea necesaria rápidamente.

3.1.2.2 Kanban

Existen distintos métodos que siguen la metodología Lean Project [9], pero se ha selecciona Kanban ya que se adapta completamente al modelo.

Durante la ejecución del proyecto se han intentado seguir las cinco propiedades clave que definen un desarrollo Kanban que indica que, si queremos obtener los mejores resultados, hay que perseguir estos objetivos [10]:

- **Visualizar el flujo de trabajo.** La transparencia y la información actualizada son críticos en generar un ambiente de confianza. Por eso la pizarra de tareas ha estado visible en todo momento.
- **Limitar el trabajo en curso.** La multitarea no existe y se ha intentado concentrar el esfuerzo en una sola tarea a la vez. Para conseguir este punto se ha limitado el número de tareas que pueden estar en cada estado.

- **Medir y optimizar el flujo.** Se ha analizado los procesos existentes, intentando no reinventar la rueda, sino optimizar lo que ya se sabe hacer bien.
- **Hacer procesos con reglas explícitas.** Las condiciones para que las tarjetas que representan esas tareas avancen de una columna a otra deben se han definido de forma clara y sencilla.
- **Gestionar el trabajo cuantitativamente.** Se han utilizado métricas que nos permitan saber, por ejemplo, cuánto tarda una tarea en completarse.

Uno de los primeros pasos al implementar Kanban es definir el tablero Kanban. El tablero tiene un fuerte efecto sobre el desarrollo del proyecto ya que aumenta la visualización y la comunicación mejorando la colaboración entre equipos.

Para poder definir dicho tablero, se han definido las columnas en las que se situarán las tareas. Estas son de dos tipos: columnas de estado y columnas de espera. Normalmente la pizarra tiene tantas columnas como estados y en ella se posicionan las tareas según el estado en el que está en cada momento. En este proyecto se han definido las siguientes columnas: pendiente, en proceso, test, hecho y cerrado.

Por otro lado, Kanban permite dividir el trabajo en tareas, normalmente identificándolas en tarjetas o fichas individuales que se colocan en una pizarra, en la que se permite crear y organizarlas, adjuntando comentarios, descripciones, enlaces o archivos.

Se ha utilizado Trello [11] como herramienta para gestión del proyecto. Es una aplicación que puede ser descargada y usada de forma gratuita, pero en nuestro caso se ha optado por usarse de forma online, ya que viene incorporada como herramienta extra dentro del propio Bitbucket. Con ella podemos seguir todos los principios Kanban e implementa las necesidades necesarias para el seguimiento del proyecto como pizarras, tareas, informes de seguimiento, etc.

Su uso ha sido fundamental tanto para llevar un seguimiento del desarrollo del proyecto y de las versiones que se han ido desarrollando, como para anotar las tareas necesarias para la tramitación, documentación y presentación del proyecto.

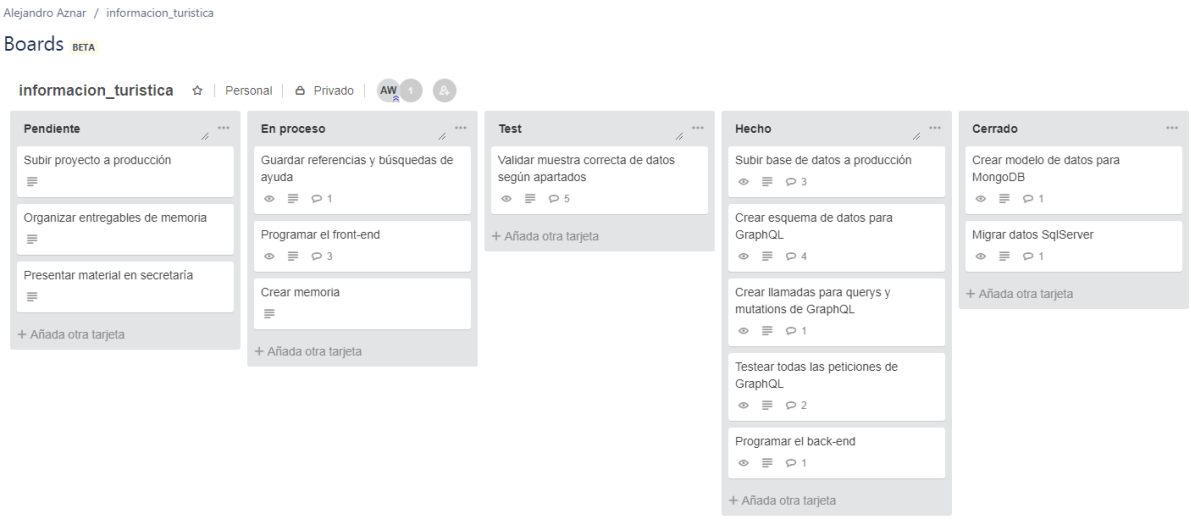


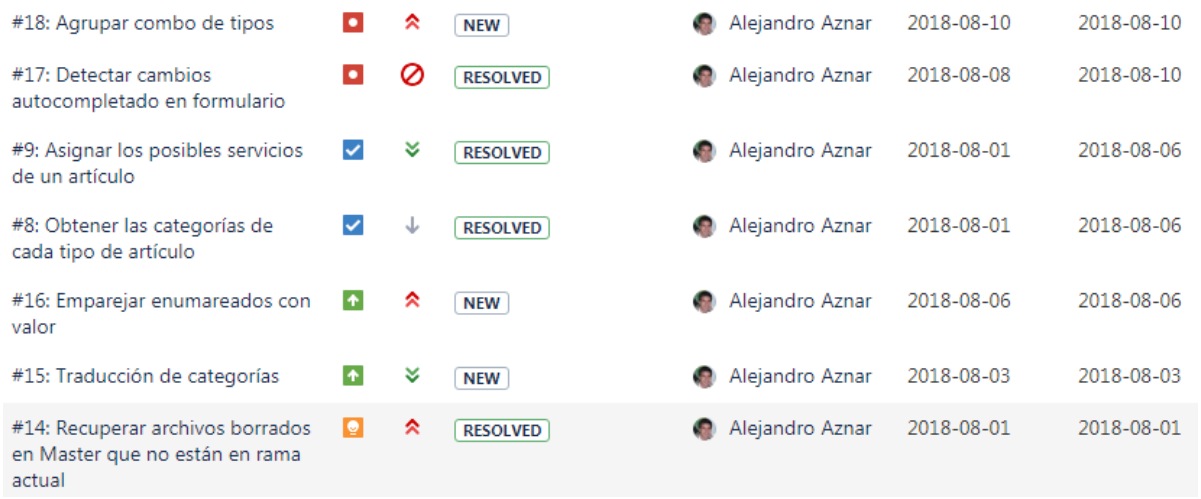
Figura 2 - Pizarra de tareas Trello durante el desarrollo del proyecto

3.1.3 Técnica de trabajo

Para todo el desarrollo de la programación se ha utilizado un sistema de incidencias para categorizar y listar todos los problemas o mejoras que nos vayamos encontrando conforme avanza el proyecto. Dentro del propio Bitbucket existe una zona “Issues” que nos permite ir creando estas incidencias. Dependiendo del grado de prioridad y del tipo, podremos hacer una clasificación sencilla e intuitiva, de forma que podamos gestionar toda la problemática de la programación más eficientemente. De esta manera, las incidencias las podemos categorizar de la siguiente forma:

3.1.3.1 Tipo de incidencia

- Bug: Son para destacar que hay que corregir algún tipo de error en la programación o mal funcionamiento de la aplicación
- Enhancement: Identifica posibles mejoras a realizar tanto en código como en funcionalidad.
- Task: Corresponden con tareas que debemos completar dentro del proyecto.
- Proposal: Están destinadas a recordarnos posibles acciones de interés solucionadas o por resolver, de forma que queden documentadas para un posible uso futuro.



#18: Agrupar combo de tipos		NEW	Alejandro Aznar	2018-08-10	2018-08-10
#17: Detectar cambios autocompletado en formulario		RESOLVED	Alejandro Aznar	2018-08-08	2018-08-10
#9: Asignar los posibles servicios de un artículo		RESOLVED	Alejandro Aznar	2018-08-01	2018-08-06
#8: Obtener las categorías de cada tipo de artículo		RESOLVED	Alejandro Aznar	2018-08-01	2018-08-06
#16: Emparejar enumareados con valor		NEW	Alejandro Aznar	2018-08-06	2018-08-06
#15: Traducción de categorías		NEW	Alejandro Aznar	2018-08-03	2018-08-03
#14: Recuperar archivos borrados en Master que no están en rama actual		RESOLVED	Alejandro Aznar	2018-08-01	2018-08-01

Figura 3 - Listado de incidencias con tipos y prioridades asociados

3.1.3.2 Prioridad de la incidencia

Clasifica el grado de importancia que tiene en el proyecto, siendo de mayor a menor:

- Blocker: La de mayor importancia. Requiere atención de forma urgente y cuanto antes.
- Critical: De gran importancia. Requiere de atención lo más pronto posible.
- Major: De cierta importancia. Es aconsejable que sean atendidas.
- Trivial: Carece de importancia, pero no estaría de más que sea tendida en algún momento.
- Minor: No posee importancia. Sólo para ser atendida si los tiempos del proyecto lo permiten.

3.2 Fases

Para completar el proyecto se ha seguido el siguiente esquema de desglose de tareas partiendo de dos partes principales: Desarrollo y Gestión.

Dentro de ambas estarán todas las fases por las que hemos ido recorriendo a lo largo del proyecto.

3.2.1 Desarrollo

Fase dedicada especialmente al análisis y programación de la aplicación. Podemos distinguir los siguientes pilares:

Planificación

- **Estudio.** Recopilación y estudio sobre gestión de proyectos y elección de la metodología a seguir.
- **Redacción.** Escritura del documento en el que se reflejarán los pasos seguidos para la realización del proyecto.

Estudio previo

- **Estudio.** Estudio previo de las metodologías y tecnologías actuales.
- **Formación.** Participación en cursos de formación de las principales tecnologías seleccionadas.
- **Prácticas.** Desarrollo de aplicaciones piloto para poner en práctica estos conocimientos.
- **Selección de tecnologías.** Elección de tecnologías y herramientas necesarias para desarrollar el proyecto.

Diseño

- **Arquitectura del sistema.** Desglose de las capas que forman la arquitectura de la aplicación.
- **Diseño de la base de datos.** Definición de valores a almacenar en la base de datos y estructuración de los mismos.
- **Identificación de los modelos de datos.** Diagramas de los distintos modelos de datos y relaciones entre los mismos.
- **Diseño de la interfaz.** Diseño de un boceto de la interfaz, para asegurar que sea intuitiva, atractiva y usable.
- **Revisión del diseño.** Refactorización, corrección y mejora del diseño del sistema.

Implementación

- **Implementación de la base de datos.** Instalación y configuración del servidor y creación y despliegue de la base de datos.
- **Desarrollo de la lógica de negocio.** Implementación de la API de la plataforma y su comunicación entre cliente-servidor por medio de GraphQL.
- **Desarrollo de interfaz.** Implementación de la capa gráfica de la aplicación.
- **Revisión de código.** Refactorización, corrección y mejora de errores surgidos durante la fase de implementación del sistema.

Pruebas

- **Diseño de pruebas**. Atendiendo a criterios de funcionalidad y usabilidad, intentando contemplar por un lado todas las posibilidades que presenta la aplicación final, y su correcta usabilidad.
- **Ejecución de las pruebas.** Ejecución de las pruebas creadas para el testeo y extracción de posibles errores y/o fallos de la aplicación.
- **Corrección y mejoras**. Corrección de errores detectados en la fase anterior.

Despliegue

- **Estudio.** Estudio de alternativas posibles.
- **Implantación.** Despliegue del proyecto en producción.
- **Manual de instalación.** Descripción detallada de la puesta en marcha.

Conclusiones

- **Puntos de mejora.** Detección y propuesta de posibles mejoras.
- **Conclusiones.** Resumen e impresiones generales y personales.

3.2.2 Gestión

Fase dedicada a la documentación y control del proyecto, sobre todo en su parte final. Se pueden distinguir los siguientes pilares:

Seguimiento

- **Revisiones.** Conjunto de comprobaciones para asegurar la coherencia del proyecto y su evolución.
- **Reuniones.** Sesiones para asegurar el correcto seguimiento y ejecución del proyecto, especialmente en el inicio y preparación de la defensa.

Documentación

- **Redacción de la memoria.** Elaboración del documento en el que se reflejarán los pasos seguidos para la realización del proyecto.
- **Redacción de anexos.** Elaboración de documentos que complementan la memoria.
- **Revisión.** Modificación y corrección de errores.

Defensa

- **Preparación.** Identificación de los puntos más importantes y preparación de la presentación.
- **Defensa.** Presentación y defensa del proyecto frente al tribunal académico.

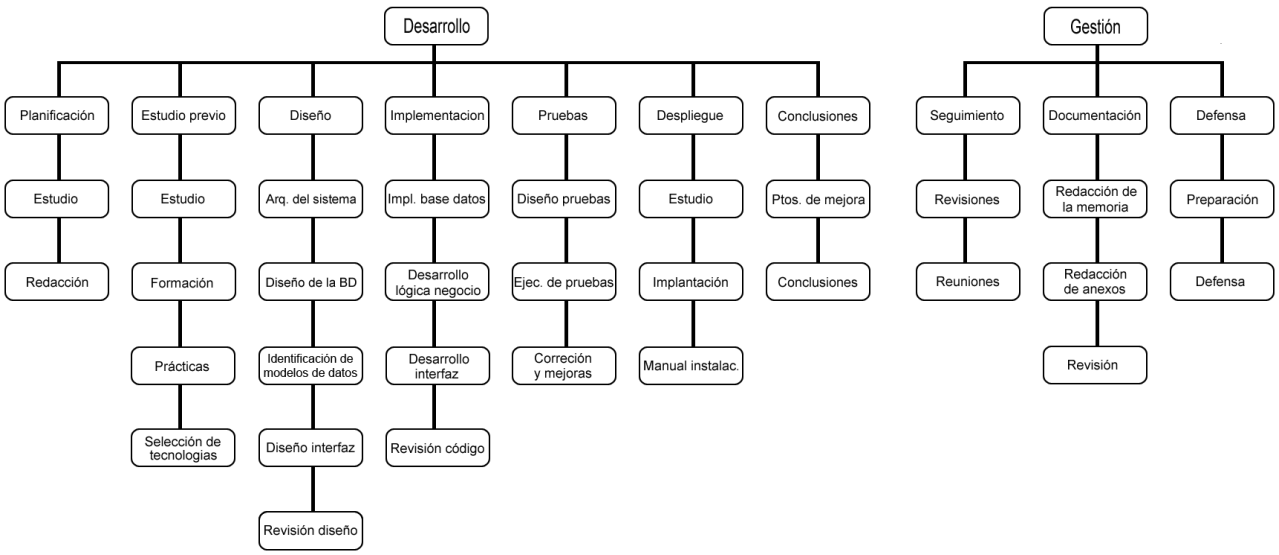


Figura 4 - Fases de trabajo

3.3 Entregables

El proyecto incluye los siguientes entregables:

- **Memoria del proyecto.** Describe todo el proceso que ha sido necesario para el desarrollo del proyecto
- **Aplicación.** Incluye el código que se ejecutará en el servidor que podrá ser accedido desde cualquier navegador.

3.4 Riesgos

Durante el desarrollo de cualquier proyecto pueden surgir complicaciones y retrasos. A continuación, se explicarán cuáles son los riesgos detectados, la probabilidad de que se produzcan, el momento previsto para su detección y el plan que se ha propuesto en cada uno de ellos en el caso de que ocurran.

3.4.1 Errores en la estimación de fechas

A pesar de haber adquirido los conocimientos teóricos necesarios durante la carrera y más de doce años de experiencia profesional en el desarrollo de aplicaciones web, existe un riesgo de estimación ya que el tiempo disponible debe ser compatibilizado con la actividad laboral y personal.

- Probabilidad alta con mayor grado al comienzo del proyecto.
- La solución propuesta es volver a planificar la estimación de fechas y aumentar el número de horas dedicadas.

3.4.2 Insuficiente conocimiento en algunas áreas

El desarrollo del proyecto necesita de muchas áreas que están fuera de la base de conocimiento y experiencia inicial, por lo que es muy probable que el estudio previo e investigación durante el proyecto necesite un tiempo muy elevado.

- Probabilidad alta a lo largo de todo el proyecto.
- Para minimizar este riesgo, se dedica una gran parte de la planificación al estudio de alternativas, pero en caso de que el tiempo dedicado a ello no sea suficiente, se propone como solución la reducción del alcance inicial.

3.4.3 Posibles problemas en el entorno de desarrollo

Se engloban todos los posibles problemas derivados de fallos técnicos, tanto en el entorno de desarrollo, como en el sistema de comunicaciones.

- Probabilidad baja a lo largo de todo el proyecto.
- Se propone la utilización de herramientas de control de versiones, copias de seguridad en la nube y en almacenamiento local para reducir el riesgo.

3.5 Plan de comunicaciones

El desarrollo del proyecto se compatibilizará en todo momento con la actividad laboral, por lo que se utilizará el correo electrónico como vía habitual de comunicación con el tutor.

Aunque no se realicen con regularidad, se establecen reuniones a lo largo del proyecto, sobre todo al inicio y tramitación inicial, definición del proyecto y en su etapa final de cara a preparar la memoria y la presentación.

4. Tecnologías y Herramientas

Antes de realizar el proyecto se ha realizado un extenso estudio sobre las principales tecnologías utilizadas actualmente en el desarrollo de aplicaciones.

En primer lugar, se analiza la evolución de la programación y la justificación del uso de JavaScript y la programación reactiva para el desarrollo del proyecto. En los siguientes apartados se describe el estudio realizado para cada las necesidades tecnológicas del proyecto, incluyendo una conclusión que explica la elección realizada en cada caso.

El inicio en el desarrollo de aplicaciones de última generación es frecuentemente confuso y difícil, debido al enorme número de bibliotecas y marcos de trabajo disponibles que están evolucionando continuamente en un espacio de tiempo muy corto. Las tecnologías de este tipo quedan obsoletas rápidamente siendo sustituidas por otras emergentes continuamente. Este estudio es, por tanto, de gran importancia para identificar las principales alternativas y determinar cuáles son las más adecuadas para el desarrollo del proyecto.

4.1 Programación

4.1.1 Antecedentes

Una ventaja significativa es que las aplicaciones web deberían funcionar igual independientemente de la versión del sistema operativo instalado en el cliente. En vez de crear clientes para cada sistema operativo, la aplicación web se escribe una vez y se ejecuta igual en todas partes. Sin embargo, hay aplicaciones inconsistentes escritas con HTML, CSS, DOM y otras especificaciones estándar para navegadores web que pueden causar problemas en el desarrollo y soporte de estas aplicaciones, principalmente debido a la falta de adhesión de los navegadores a estándares web. Adicionalmente, la posibilidad de los usuarios de personalizar muchas de las características de la interfaz puede interferir con la consistencia de las aplicaciones web.

Las ventajas de las aplicaciones web son evidentes, pero hasta la popularización de HTML5, se necesitaba la adopción de tecnologías alternativas

para el desarrollo de aplicaciones complejas. Esta solución, conocida como aplicación de Internet enriquecida, puede ofrecer la mayoría de las características de las aplicaciones de escritorio tradicionales en un navegador. El problema es que necesitan la instalación de complementos o la ejecución de una máquina virtual para agregar las características adicionales.

4.1.2 HTML 5

HTML es un elemento de construcción de contenido web, un lenguaje de marcas que solamente define el contenido y estructura de la página, no su funcionalidad. Es un lenguaje interpretado por el navegador, encargado de mostrar el contenido web tal y como lo especifica el HTML. Fue creado en 1990 y es un estándar internacional, otras características es que está regulado por el W3C (World Wide Web Consortium) y el WHATWG (Web Hypertext Application Technology Working Group.

Algunas de las características de HTML5 son:

- Facilita el desarrollo de componentes gracias.
- Etiquetas sintácticas.
- Etiquetas multimedia.
- Nuevas capacidades en los formularios.
- Geolocalización.

4.1.3 CSS3

Es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML2 (y por extensión en XHTML). El W3C es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

Algunas ventajas de utilizar CSS son:

- Control centralizado de la presentación de un sitio web completo con lo que se agiliza de forma considerable la actualización del mismo.

- Separación del contenido de la presentación, lo que facilita al creador, diseñador, usuario o dispositivo electrónico que muestre la página, la modificación de la visualización del documento sin alterar el contenido del mismo, sólo modificando algunos parámetros del CSS.
- Optimización del ancho de banda de la conexión, pues pueden definirse los mismos estilos para muchos elementos con un solo selector; o porque un mismo archivo CSS puede servir para una multitud de documentos.
- Mejora en la accesibilidad del documento, pues con el uso del CSS se evitan antiguas prácticas necesarias para el control del diseño (como las tablas), y que iban en perjuicio de ciertos usos de los documentos, por parte de navegadores orientados a personas discapacitadas.

4.1.4 JavaScript

JavaScript es un lenguaje ligero, orientado a objetos e interpretado. En un primer momento se ha encargado de dar dinamismo a las webs. Hoy en día JavaScript se está utilizando para otros propósitos, como podemos ver en NodeJS o Apache CouchDB.

Con la llegada de los nuevos navegadores HTML5, se demuestra que son capaces de visualizar páginas web cada vez más potentes y espectaculares. Al mismo tiempo han ido surgiendo nuevos marcos de trabajo en JavaScript, donde el desarrollo de aplicaciones web del lado del cliente está siendo cada vez más rápido, complejo y con una apariencia y capacidades cada vez más similares a las aplicaciones de escritorio.

Para dar una mejor experiencia de uso al usuario, la tendencia actual es trasladar la carga de trabajo del servidor hacia el navegador del cliente, y ejecutar allí totalmente la aplicación web con muy poca comunicación con el servidor. A estas aplicaciones se las conoce como SPA o aplicación en una sola página. Cuando una aplicación de este tipo inicia la ejecución, se descargan todos los archivos HTML, JavaScript y CSS que forman el esqueleto de la aplicación web, y el resto de los recursos necesarios se van solicitando posteriormente bajo demanda según el usuario vaya interactuando con la aplicación.

La elección de JavaScript [12] [13] en el lado cliente es clara, al ser uno de los lenguajes más flexibles y populares actualmente, por su continua actualización y

el inmenso contenido creado por los usuarios. Gracias a sus características, permite a priori navegar por webs de elevado contenido de información de manera rápida y fluida. Algunas de estas características son, principalmente:

- Lenguaje dinámico.
- Soporta programación funcional, orientada a objetos e imperativa.
- Asignación de tipos dinámica.
- Lenguaje interpretado.

Ventajas

- Completa interacción con los elementos HTML y con CSS. Mediante JavaScript es posible manipular las propiedades de los objetos y actualizar, establecer o eliminar ciertos valores para cambiar su comportamiento.
- JavaScript es un lenguaje ligero, es decir, el peso de los archivos contenidos en la aplicación es bajo, lo que hace que la web cargue muy rápida, y los eventos asociados a ella se reproduzcan de forma fluida.
- Las aplicaciones webs desarrolladas en JavaScript funcionan en cualquier plataforma (IOs, Windows, Linux, Android, …) y sobre cualquier navegador (Google Chrome, Mozilla Firefox, Internet Explorer, …)
- JavaScript es un lenguaje multiplataforma, por lo que no condiciona al desarrollador la elección de un determinado sistema o entorno de trabajo.

Desventajas

- Uno de los grandes problemas de los desarrolladores no tiene que ver con el desarrollo en sí, sino con la incompatibilidad actual que existe entre los navegadores. Una misma página web puede ejecutarse correcta y eficientemente en un navegador, pero en otros no, aunque este problema puede ser resuelto parcialmente con el uso algunas de bibliotecas auxiliares.
- Debido a que JavaScript se ejecuta del lado del cliente y es un lenguaje interpretado, se puede acceder al código con facilidad. Existen técnicas de ofuscación del código, aunque siguen sin ser realmente seguras, por

lo que exige al desarrollador un especial esfuerzo en cuidar la seguridad de la aplicación.

- La ejecución puede requerir bastante memoria, especialmente al ejecutar operaciones matemáticas complejas.

4.1.5 Programación reactiva

En los últimos tiempos están sucediendo cambios en el ámbito de la computación y están apareciendo nuevos conceptos y paradigmas que intentan dar respuesta a nuevas necesidades y la tecnológica actual. Uno de los términos en auge es la programación reactiva, que se basa en un conjunto de patrones y técnicas que están ganando mucha importancia en los últimos años.

Hace una década, las aplicaciones más grandes contaban con un número de servidores pequeños, sus tiempos de respuesta no eran tan exigentes, los servidores podían parar horas para ser mantenidos y el volumen de datos era muy inferior al actual.

El objetivo de la programación reactiva es que podamos, de una manera muy fácil en nuestro lenguaje de preferencia, expresar flujos de datos (ya sean estáticos o dinámicos) que reaccionen a cambios en ciertas condiciones y propagar estos cambios a todos los elementos relacionados.

4.1.6 GraphQL vs RESTful

La necesidad de avanzar más rápido en productos cada vez más complejos ha empujado un cambio en la forma en que interactuamos con las APIs. Aquí es dónde surge GraphQL, un fuerte candidato a sustituir a REST [14] [15], sobre todo en el ecosistema de APIs para aplicaciones móviles.

El problema que nos presenta actualmente RESTful [16] es que, para obtener un dato en concreto, casi seguramente tendremos que descargar más información de la necesaria, y con ello el consiguiente exceso de tráfico adicional para el usuario. Es decir, en la práctica real, en RESTful utilizaríamos una URI para leer o escribir un único recurso, ya sea, por ejemplo, un producto, una persona, un post o un comentario. Si necesitamos trabajar con múltiples recursos como un listado de posts con un listado de comentarios, necesitaríamos manejar múltiples endpoint y encadenar distintas llamadas, a veces de forma secuencial. Cuando

hacemos una petición, no recibimos simplemente la información que necesitamos, sino todo el conjunto de datos relativos al recurso alojado en esa URI. Algo bastante complicado de gestionar y costoso en recursos cuando el 90% de los datos procedentes de cada recurso son innecesarios. Por ejemplo, en la *Figura 5* podemos ver cómo obtenemos únicamente los datos que solicitamos, en este caso todos los títulos de las películas de Star Wars, su director correspondiente, y el nombre de todas las especies que aparecen en cada una de ellas [18].

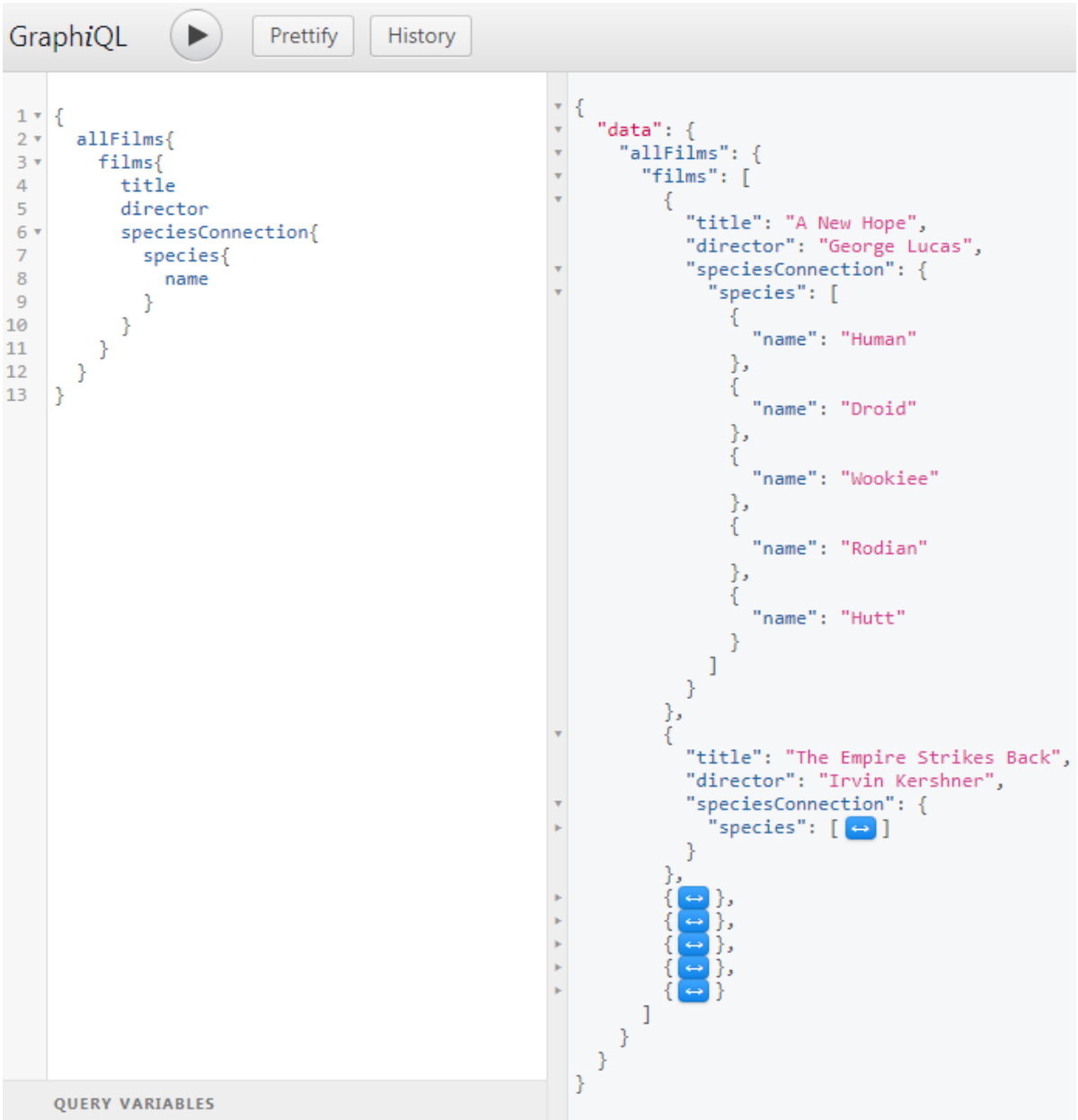


Figura 5 - Ejemplo de petición en GraphQL

GraphQL [17] es una de las alternativas que han surgido para solucionar la mayor parte de estos problemas. Con REST, no podemos elegir los datos que queremos recibir en el JSON de respuestas; en cambio en GraphQL podemos elegir exactamente lo que necesitamos [18].

Esta herramienta fue creada por Facebook en 2013 como lenguaje de consultas, pero no se hizo código abierto hasta 2015. Fue creado netamente para la comunicación cliente-servidor.

GraphQL no es una librería o framework: es una especificación de cómo implementarlo en cualquier lenguaje. Además, existen implementaciones ya creadas en lenguajes como JavaScript, Ruby, Python, Scala, Java, Clojure, Go, PHP, .NET, etc. [30] También existen clientes para consumir un API GraphQL desde JS, iOS, Android, React [31], Angular, entre otras.

Mientras que en REST realizábamos peticiones HTTP usando métodos tales como *GET*, *POST*, *PUT* o *DELETE* por ejemplo, en GraphQL sólo usaremos el método *POST*. Dependiendo si queremos datos o manipularlos, tan sólo deberemos de poner en el cuerpo de la consulta alguna de estas opciones:

- **Query.** Sería el equivalente *GET* en REST. Sólo obtiene los datos solicitados.
- **Mutation.** Además de obtener los datos que precisemos, nos permite realizar modificaciones sobre ellos (crear, modificar, eliminar, …)
- **Suscription.** No está implementado por todas las librerías del lado servidor, y que es reciente el que se volviera parte oficial de la especificación de GraphQL. Lo que nos permiten las suscripciones es, como su nombre dice, suscribirnos a cambios que ocurran en el servidor.

Para toda API de GraphQL debemos desarrollar un esquema de datos (**schema**), donde podremos definir el tipo de los datos, o la forma de obtener e interactuar con ellos. Pero con todo ello no basta. Debemos decirle cómo interactuar con la base de datos para obtener la información, y eso se consigue con los llamados **resolvers**. Son funciones que se encargan de procesar cada posible consulta, mutación o suscripción de nuestra API y de responder con los datos necesarios, según la definición de nuestro esquema.

4.1.6.1 GraphiQL

Es el propio IDE (Integrated Development Environment) creado por Facebook. Este IDE funciona mediante web, mostrándose, normalmente, en la URL de nuestro API cuando entramos desde un navegador. Este se conecta con nuestros esquemas de datos para mostrarnos documentación del API y nos deja probarlo, dando sugerencias de autocomplete y mostrándonos las respuestas a las distintas peticiones, mutaciones y suscripciones. Puede verse un ejemplo en la *Figura 5*.

4.1.7 Conclusión

Hoy en día los clientes móviles (sobre todo) necesitan tener más poder de decisión sobre los datos que necesitan consumir. Esto les provee de mayor independencia sobre los servicios de datos. Las peticiones son enviadas desde el cliente, lo que simplifica la retrocompatibilidad explícita.

Como se ha descrito, la plataforma debe reaccionar a la interacción de los usuarios ante la búsqueda de información, por lo que ésta debe actualizarse en tiempo real en función de los eventos que se produzcan. Esto, unido a una óptima obtención de los datos entre cliente y servidor justifican la elección del paradigma de la programación reactiva y el uso de GraphQL, que encaja perfectamente con el funcionamiento y las necesidades de la aplicación.

4.2 Marcos de trabajo

Si bien JavaScript se considera un lenguaje sencillo de aprender, no se plantea la opción de partir desde cero ya que existen numerosos marcos de trabajo JavaScript que facilitan enormemente el desarrollo de aplicaciones.

4.2.1 Marcos de trabajo isomorfos: NodeJS

Frente a esta tendencia en la que la capa de acceso a datos se desarrolla con otros lenguajes mientras que toda la aplicación es generada y administrada a través de un marco de trabajo MV- en JavaScript, existe otro enfoque más reciente que elimina la redundancia y simplifica el desarrollo.

La ventaja de utilizar un marco de trabajo isomorfo es que se utiliza el mismo lenguaje de programación, tanto para el lado del cliente como para el lado del servidor.

En el lado del cliente no hay problema, porque los navegadores web llevan mucho tiempo con la facultad de ejecutar JavaScript. Y en el lado del servidor, gracias a la utilización de NodeJS [28] como servidor web, se puede ejecutar código en JavaScript.

NodeJS está basado en el lenguaje de programación ECMAScript, asíncrono, basado en una arquitectura orientada a eventos sobre el motor V8 de Google. Soporta protocolos TCP, DNS y HTTP, y fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables como, por ejemplo, servidores web.

NodeJS, como entorno en tiempo de ejecución multiplataforma de código abierto, es el socio ideal para el desarrollo de aplicaciones web, donde se implementa fácilmente la comunicación bidireccional, ya que posibilita el desarrollo de aplicaciones web que se comunican con todos los clientes en tiempo real.

La arquitectura actual se ha ido volviendo más compleja en el lado cliente, empezando a ser muy parecida a la del lado servidor. La elegancia de la propuesta de arquitectura isomorfa se basa en utilizar JavaScript en los dos entornos, de forma que la arquitectura, código y datos se puedan compartir haciendo que se ejecute en los dos lados indistintamente.

4.2.2 Marcos de trabajo en el lado cliente: React

Los marcos de trabajo implementados para este tipo de aplicaciones web están basados en el paradigma MVC, o Modelo-Vista-Controlador, del lado del cliente. Los más utilizados suelen ser AngularJS, Backbone.js y Ember.js. Estos marcos de trabajo nos permiten trasladar gran parte de la lógica de la vista, modelo y controlador de la aplicación web, hacia el al lado del cliente, ganando con ello en velocidad, por lo que proporcionamos una mejor experiencia de uso al usuario de nuestra aplicación web.

La relación de la aplicación web, en vez de realizarse con el servidor, se hace con la API encargada de gestionar la lógica de la misma. El servidor queda

relegado para descargar, el esqueleto de la página inicial de la aplicación y los archivos adicionales, una vez inicializada la aplicación web en el lado del cliente.

El problema es que estos marcos de trabajo trabajan sólo en el lado del cliente, por lo que partes de la lógica en JavaScript que ya existe en el lado del cliente, tenemos que volverlas a programar otra vez en el lado del servidor, y en otro lenguaje distinto, traduciendo estas partes ya sea al lenguaje PHP, Java, Python, Ruby, etc.

El modelo más popular en estos momentos apuesta por la separación de responsabilidades entre Modelo, Vista y Controlador. Prácticamente todos los marcos de trabajo web han implementado este enfoque de una forma u otra, (incluidos como Struts en Java, ASP.NET en Microsoft o Laravel en PHP). El lado cliente es sencillo y la arquitectura más compleja está solo en el lado servidor.

Con la llegada de las aplicaciones en una sola página, el lado cliente comienza a tener más peso en los desarrollos y organiza mejor el código de JavaScript. La arquitectura SPA MVC propone una arquitectura en el lado cliente equivalente al lado servidor. Aparecen los primeros marcos de trabajo MVC de cliente como Backbone.js que permiten dividir las responsabilidades de la misma forma que en el servidor.

Para el desarrollo de este proyecto se han evaluado marcos de trabajo como AngularJS que, aunque son válidos para el desarrollo de aplicaciones, son soluciones parciales que utilizan JavaScript sólo en el lado cliente.

Finalmente se ha optado por usar React frente a AngularJS debido a que no es un framework, sino una librería que pesa muchísimo menos y cuya finalidad es ideal para nuestros propósitos, el poder gestionar la vista de nuestra aplicación de forma reactiva, separando su funcionalidad por componentes. A esto hay que sumarle el gran respaldo que hay actualmente en toda la comunidad de programadores, y el que está en pleno auge ascendente de uso y popularidad, quizás también por el hecho de ser una herramienta de un gran coloso como es Facebook.

React es una librería que proviene de Facebook. Es de software libre, y a partir de su liberación una creciente comunidad de desarrolladores la está usando. Se crea en base a unas necesidades, generadas por el propio desarrollo de la web

de la popular red social. En Facebook necesitaban herramientas para un desarrollo rápido pero focalizadas en un mayor rendimiento que otras alternativas existentes en el mercado.

Detectaron que el típico marco de binding y doble binding ralentizaba un poco su aplicación, debido a la cantidad de conexiones entre las vistas y los datos. Como respuesta crearon una nueva dinámica de funcionamiento, en la que optimizaron la forma como las vistas se pintaban frente al cambio en los datos de la aplicación.

A partir de ahí la probaron en su red social con resultados positivos y luego en Instagram, también propiedad de Facebook, con éxito. Más adelante, alentados por los positivos resultados en el rendimiento de React, muchas otras aplicaciones web de primer nivel la fueron adoptando. BBC, Airbnb, Netflix, Dropbox y un largo etc.

Como ya hemos visto anteriormente, React es una librería focalizada en el desarrollo de interfaces de usuario que permite crear aplicaciones web y el isomorfismo. Esto es que, con el mismo código somos capaces de renderizar tanto en el cliente como el servidor. Por tanto, cuando llega un buscador como Google, con la misma base de código se le puede entregar el HTML con el contenido ya pintado por pantalla, lo que lleva a que una aplicación React sea capaz de posicionarse tan bien como una aplicación web tradicional que renderice del lado del servidor.

Algunas de sus principales características son [29]:

- **Composición de componentes.** Así como en programación funcional se pasan funciones como parámetros para resolver problemas más complejos, creando lo que se conoce como composición funcional, en ReactJS podemos aplicar este mismo patrón mediante la composición de componentes

  Las aplicaciones se realizan con la composición de varios componentes. Estos componentes encapsulan un comportamiento, una vista y un estado. Pueden ser muy complejos, pero es algo de lo que no necesitamos preocuparnos cuando estamos desarrollando la aplicación,

porque el comportamiento queda dentro del componente y no necesitamos complicarnos por él una vez se ha realizado.

Este modelo de trabajo tiene varias ventajas, como la facilidad de mantenimiento, depuración, escalabilidad, etc.

- **Desarrollo Declarativo vs Imperativo.** En la experiencia de desarrollo con librerías más sencillas como jQuery realizamos un estilo de programación imperativo. En ese estilo se realizan scripts que paso por paso tienen que informar sobre qué acciones o cambios en el DOM se deben realizar. Hay que ser muy concisos en esas acciones, especificando con detalle cada uno de los cambios que se quieren realizar. La forma imperativa de declarar nos obliga a escribir mucho código, porque cada pequeño cambio se debe definir en un script y cuando el cambio puede ser provocado desde muchos sitios, cuando agregamos eventos, el código comienza a ser poco mantenible.

  Sin embargo, el estilo de React es más declarativo, en el que nosotros contamos con un estado de la aplicación y sus componentes reaccionan ante el cambio de ese estado. Los componentes tienen una funcionalidad dada y cuando cambia una de sus propiedades ellos producen un cambio. En el código de nuestra aplicación tendremos ese componente, y en él se declarará de donde vienen los datos que él necesita para realizar su comportamiento. Podremos usarlo tantas veces como queramos declarando que lo queremos usar y declarando también los datos que él necesita para funcionar.

- **Flujo de datos unidireccional.** Ésta es otra de las cosas que facilita React, aunque no es exclusivo. En este modelo de funcionamiento, los componentes de orden superior propagan datos a los componentes de orden inferior. Los de orden inferior trabajarán con esos datos y cuando cambia su estado podrán propagar eventos hacia los componentes de orden superior para actualizar sus estados.

  Este flujo tiende a ser unidireccional, pero entre componentes hermanos muchas veces se hace más cómodo que sea bidireccional y también se puede hacer dentro de React. Sin embargo, si tratamos siempre de mantener el patrón de funcionamiento unidireccional, nos facilitará mucho el mantenimiento de la aplicación y su depuración.

- **Performance gracias al DOM Virtual.** El desempeño de React es muy alto, gracias a su funcionamiento. Nos referimos al desempeño a la hora del renderizado de la aplicación. Esto se consigue por medio del DOM Virtual. No es que React no opere con el DOM real del navegador, pero sus operaciones las realiza antes sobre el DOM Virtual, que es mucho más rápido.

  El DOM Virtual está cargado en memoria y gracias a la herramienta que diferenciación entre él y el real, el DOM del navegador se actualiza. El resultado es que estas operaciones permiten actualizaciones de hasta 60 frames por segundo, lo que producen aplicaciones muy fluidas, con movimientos suavizados.
- **Isomorfismo.** Es la capacidad de ejecutar el código tanto en el cliente como el servidor. También se conoce como "JavaScript Universal". Sirve principalmente para solucionar problemas de posicionamiento tradicionales de las aplicaciones JavaScript.
- **Elementos y JSX.** ReactJS no retorna HTML. El código embebido dentro de JavaScript parece HTML, pero realmente es JSX. Son como funciones JavaScript, pero expresadas mediante una sintaxis propia de React llamada JSX. Lo que produce son elementos en memoria y no elementos del DOM tradicional, con lo cual las funciones no ocupan tiempo en producir pesados objetos del navegador sino simplemente elementos de un DOM virtual. Todo esto, como hemos dicho, es mucho más rápido.
- **Componentes con y sin estado.** React permite crear componentes de diversas maneras, pero hay una diferencia entre componentes con y sin estado.

  Los componentes “stateless” son los componentes que no tienen estado, digamos que no guardan en su memoria datos. Eso no quiere decir que no puedan recibir valores de propiedades, pero esas propiedades siempre las llevarán a las vistas sin producir un estado dentro del componente. Estos componentes sin estado se pueden escribir con una sencilla función que retorna el JSX que el componente debe representar en la página.

  Los componentes “statefull” son un poco más complejos, porque son capaces de guardar un estado y mantienen lógica de negocio

generalmente. Su principal diferencia es que se escriben en el código de una manera más compleja, generalmente por medio de una clase ES6 (JavaScript con ECMAScript 2015), en la que podemos tener atributos y métodos para realizar todo tipo de operaciones. Los componentes statefull, con estado, necesitan tener un método *render()* que se encarga de devolver el JSX que usar para representarlo en la página.

Podemos decir que los componentes con estado se usan para resolver problemas mayores, con lógica de negocio, mientras que los stateless se usan más para interfaces de usuario.

- **Ciclo de vida de los componentes.** React implementa un ciclo de vida para los componentes. Son métodos que se ejecutan cuando pasan cosas comunes con el componente, que nos permiten suscribir acciones cuando se produce una inicialización, se recibe la devolución de una promesa, etc.

4.2.3 Conclusión

Con este nuevo paradigma, partes lógicas de la aplicación se puede compartir en el lado cliente y servidor. Esto significa abrir un gran número opciones y optimizar el rendimiento en el desarrollo. Crear aplicaciones con esta arquitectura puede ser algo muy complejo al principio por ser diferente, y más si cabe cuando la experiencia previa respecto a conocimientos y filosofía de trabajo en estas nuevas tecnologías es completamente nula; pero al final todo resulta más sencillo y fácil de mantener.

4.3 Bases de datos

Inicialmente disponemos de una extensa base de datos en SQL Server propiedad de la Excma. Diputación de Málaga. Por motivos laborales actuales, y con la finalidad de mejorar en un futuro la actual web de información turística de la propia Diputación de Málaga [58], se nos ha ofrecido la oportunidad de poder manipular estos datos con fines académicos.

Para ello, en la implementación del proyecto se han analizado también diferentes bases de datos que permitan su integración o estén directamente desarrolladas en JavaScript.

El estudio se ha centrado en bases de datos NoSQL [19] que se refiere a un tipo de base de datos que no se ajusta al modelo de bases de datos relaciones. En ellas no se garantiza la propiedad ACID (Atomicity, Consistency, Isolation, Durability) ya que priorizan otro tipo de aspectos como la escalabilidad, el rendimiento o la flexibilidad de su estructura.

4.3.1 Bases de datos NoSQL

Entre sus características, se puede destacar las siguientes ventajas con respecto a otro tipo de bases de datos:

- Permiten manejar volúmenes de información mucho más grandes con una velocidad de acceso rápida.
- Facilitan la escalabilidad horizontal ya que son fáciles de usar en grupos de servidores con balanceo de carga.
- Poseen estructuras de datos flexibles permitiendo añadir nuevos campos de forma dinámica o efectuar una migración sin tener que ser reiniciadas o paradas.

Por otra parte, las grandes diferencias que existen entre las bases de datos relacionales y las no relacionales son:

- No usan SQL como lenguajes para realizar las consultas. La mayoría de las bases de datos NoSQL tienen formas distintas de realizar las consultas por medio de algún lenguaje de apoyo, por ejemplo, MongoDB utiliza JSON y Cassandra utiliza el lenguaje CQL.
- Usan una arquitectura distribuida de forma nativa. Con este tipo de arquitectura se nos permite tener la información de forma compartida entre varias máquinas.
- No permiten operaciones de JOIN. La mayoría de las bases de datos NoSQL al manejar grandes volúmenes de información la realización de un JOIN es algo costoso por lo cual la forma de hacerlo es mediante el software.

No existe un estándar para todas. La gran diferencia con las bases de datos relaciones radica en que existen varias formas de almacenar la información (orientadas a clave-valor, a columnas, a documentos o a grafos).

4.3.2 Bases de datos analizadas

4.3.2.1 MongoDB

MongoDB es el sistema de base de datos NoSQL más extendido actualmente. Se basa en el concepto de documento, que se pueden agrupar en colecciones, que son el equivalente a las tablas en una base de datos relacional (sólo que las colecciones pueden almacenar documentos con muy diferentes formatos, en lugar de estar sometidos a un esquema fijo). Se pueden crear índices para algunos atributos de los documentos, de modo que MongoDB mantendrá una estructura interna eficiente para el acceso a la información por los contenidos de estos atributos.

Los distintos documentos se almacenan en formato BSON (Binary JSON), que es una versión modificada de JSON que permite búsquedas rápidas de datos, y es de uso común en la programación web, sobre todo en JavaScript. Esto resulta especialmente interesante para proyectos web con NodeJS. MongoDB [20] está escrito en C++, por lo que es bastante rápido a la hora de ejecutar sus tareas. Además, está licenciado como GNU AGPL 3.0, de modo que se trata de un software de licencia libre.

4.3.2.2 CouchDB

CouchDB [21] es una base de datos pensada específicamente para aplicaciones web, almacena los datos en formato JSON y puede acceder a los datos directamente por HTTP a través de una API. Utiliza JavaScript para indexar, combinar o transformar los documentos. Funciona eficientemente en aplicaciones móviles. Está bastante centrada en la disponibilidad y permite tener los datos distribuidos.

4.3.2.3 Cassandra

Apache Cassandra [22] es una de las mejores opciones cuando la escalabilidad, la alta disponibilidad y la integridad de los datos son las prioridades de un proyecto.

4.3.2.4 Conclusión

La perfecta integración con NodeJS y React, sumado con lo ligera que es, la facilidad de su uso y el que permita una sencilla escalabilidad, ha hecho que se haya elegido MongoDB en este proyecto. Ésta se ha convertido también en la base de datos NoSQL más extendida en la comunidad informática, por lo que su aprendizaje tiene un interés adicional frente a las anteriores al disponer de más recursos de ayuda.

4.3.3 Formatos de datos

JSON Es un formato de texto ligero para el intercambio de datos que se basa en un subconjunto de la notación literal de objetos de JavaScript. Debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

4.4 Aplicaciones

4.4.1 Bases de datos

4.4.1.1 SQL Server 2014 Management Studio

Nos servimos de este programa gratuito para poder migrar los datos originales mediante consultas SQL. Una vez devuelto los datos en un determinado formato y tras limpiarlos, tan solo había que copiar y pegar esa información en su colección correspondiente, dentro del programa Robomongo.

4.4.1.2 Robomongo

También conocido como Robo 3T [27], es una aplicación multiplataforma que se instala de forma rápida y sencilla en nuestro equipo. Con ella podemos acceder a un interfaz mucho más amigable e intuitiva para administrar gráficamente nuestra

base de datos. De este modo nos evitamos tener que hacer incómodas consultas desde línea de comandos de nuestro terminal para tratar directamente con los datos de MongoDB. Por esta razón, Robomongo se ha convertido en un importante aliado para los usuarios de esta base de datos, pues podemos tener rápido acceso a las *colecciones* de nuestra base de datos y a todos sus *documentos*.

4.4.2 Gestión

4.4.2.1 Git

Se trata de un sistema de control de versiones distribuido diseñado para manejar todo tipo de proyectos con eficiencia.

4.4.2.2 Bitbucket

Es un servicio de alojamiento basado similar a GitHub, para los proyectos que utilizan el sistema de control de revisiones Mercurial y Git. Se ofrece cuentas gratuitas, como la utilizada para este proyecto, con un número ilimitado de repositorios privados.

4.4.2.3 Trello

Es la aplicación de referencia actual en gestión de proyectos ágiles. El aspecto visual ofrece una interfaz sencilla, con un uso bastante intuitivo que permite arrastrar tarjetas entre los diferentes estados o columnas y crear elementos en cualquier área de forma rápida. Es perfecto para cualquier persona que necesita un tablero Kanban.

4.4.3 Documentación

LibreOffice

LibreOffice es un paquete de software de oficina libre y de código abierto desarrollado por The Document Foundation. Se creó como bifurcación de OpenOffice en 2010.

Cuenta con un procesador de texto, un editor de hojas de cálculo, un gestor de presentaciones, un gestor de bases de datos, un editor de gráficos vectoriales y un editor de fórmulas matemáticas.

El formato de archivo propio de LibreOffice es OpenDocument, un formato estándar y abierto que está siendo adoptado por gobiernos de todo el mundo como formato de archivo obligatorio para la publicación y aceptación de documentos. Puede abrir y guardar documentos en muchos otros formatos, incluyendo aquellos utilizados en varias versiones de Microsoft Office.

Está diseñado para ser compatible con los principales paquetes ofimáticos, incluyendo Microsoft Office, aunque algunas características son manejadas de forma diferente o no son compatibles. Está disponible en más de cien idiomas y casi todos los sistemas operativos y es la suite ofimática por defecto en las distribuciones Linux más populares.

4.4.4 Desarrollo

Visual Studio Code

Aunque inicialmente se empezó el proyecto usando Atom, finalmente nos decantamos por elegir Visual Studio Code. Es un editor [33] gratuito para macOS, Linux, y Windows con soporte para complementos escrito en NodeJS y desarrollado por Microsoft. Incluye soporte para la depuración, control integrado de Git [34], resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que permite cambiar el tema del editor, los atajos de teclado y las preferencias.

Npm

Es el gestor de paquetes por defecto para NodeJS. Se ejecuta desde la línea de comandos y maneja las dependencias para una aplicación. Además, permite a los desarrolladores instalar aplicaciones NodeJS que se encuentran en el repositorio. Al igual que muchas herramientas usadas en este proyecto, está escrito enteramente en JavaScript.

4.4.5 Depuración

Firebug

Es una extensión de Firefox [36] creada y diseñada especialmente para desarrolladores y programadores web. Es un paquete de utilidades con el que se puede analizar (revisar velocidad de carga, estructura DOM), editar, monitorizar y

depurar el código fuente, CSS, HTML y JavaScript de una página web de manera instantánea.

Se ha utilizado hasta noviembre del 2017 ya que, con el lanzamiento de Firefox Quantum, todas las capacidades de Firebug ahora están presentes en las herramientas actuales de desarrollo de Firefox.

Chrome DevTools

Es un conjunto de herramientas [37] de creación web y depuración integrado en Google Chrome, de alguna forma similar a Firebug.

4.4.6 Ejecución

Heroku

Heroku [24] es una plataforma como servicio de computación en la nube que soporta distintos lenguajes de programación y plataformas, incluida NodeJS.

Navegadores

Para la ejecución de la aplicación se han utilizado principalmente Firefox Developer Edition, que es una versión de Firefox especial para desarrolladores web que incluye herramientas y opciones no disponibles en la versión normal de Firefox.

También se ha verificado su funcionamiento en Chrome.

4.5 Alojamiento

Actualmente hay muchas infraestructuras y plataformas que permitan alojar una aplicación hecha con NodeJS. Y estos son algunos de los alojamientos analizados.

4.5.1 Alojamientos valorados

Digital Ocean

Es el más básico de todos [23] y requiere una configuración algo más laboriosa. No posee cuentas gratuitas para su almacenaje.

Heroku

Heroku es una plataforma de servicios genérica por lo que admite NodeJS y React sin problemas. La arquitectura de Heroku se basa en dynos, que son las unidades que proveen capacidad de cómputo dentro de la plataforma. Están basados en contenedores Linux aislados del resto, por lo que los comandos que se ejecutan y los archivos que se almacenan no afectan a los otros.

Su principal ventaja es que tiene una opción de prueba que permite instalar aplicaciones sin costes iniciales [24], aunque su configuración requiera algo más de dedicación.

Openshift

Es una plataforma creada por RedHat que permite desplegar aplicaciones en distintos lenguajes como PHP, Ruby o el propio NodeJS. Existe la posibilidad de crear y ejecutar aplicaciones en su nube pública [25], de forma que cualquiera puede crearse una cuenta gratuita y desplegar su aplicación.

NodeChef

Es una plataforma dedicada para aplicaciones NodeJS y MongoDB que cuenta con un único comando de despliegue [26] y, a diferencia de otros, incluye una capa de base de datos. Sólo existe en versión de pago, pero el precio incluye el aprovisionamiento automático de certificados SSL, la instalación y renovación, la administración de dominios comodín y el almacenamiento en caché de activos estáticos.

4.5.2 Conclusión

De entre todas las opciones analizadas, se ha elegido Heroku, ya que es principalmente la única que permite desplegar aplicaciones en para comprobar su funcionamiento sin coste adicional, y la que posee mayor respaldo de información por la comunidad informática ante posibles problemas en su uso.

Las principales características que ofrece esta plataforma son:

- **Elasticidad y crecimiento.** La cantidad de dynos asignados a una aplicación se puede cambiar en cualquier momento a través de la línea de comandos o el cuadro de mandos.

- **Tamaño.** Heroku ofrece diferentes tipos de dynos con diferentes capacidades de procesamiento y memoria.
- **Encaminado.** Internamente los encaminadores realizan un seguimiento de la ubicación de los dynos que estén ejecutándose y dirigen el tráfico de la misma.
- **Seguimiento.** Existe un manejador de dynos, el cual monitorea de forma continua los dynos que se estén ejecutando. En caso de una se elimina y se crea nuevamente.
- **Distribución y redundancia.** Como dynos se encuentran aislados uno de otro, en caso de fallos en la infraestructura interna de alguno de ellos, los otros dynos no se ven afectados.

5. Desarrollo

Este capítulo describe el proceso que se ha seguido para el desarrollo de la aplicación, teniendo como principal objetivo crear un código limpio, legible, claro, fácil de extender y adaptable a futuros cambios. [37].

En primer lugar, explicamos cuáles son los preparativos iniciales de configuración del entorno y un breve resumen de la instalación y paquetes que ha sido necesarios para el desarrollo.

En los siguientes apartados se describe una visión global de la aplicación, explicando cuál es su estructura y cómo se han organizado los archivos en la aplicación.

Por último, en los siguientes apartados, se explica con más detalle el proceso de desarrollo para cada uno de los apartados, como base de datos, comunicaciones, diseño, etc.

5.1 Preparativos

5.1.1 Migración de datos originales

Gracias a la Excma. Diputación de Málaga, disponemos originalmente de una amplia base de datos en SQL Server. Para poder usar estos datos en MongoDB debemos migrar los datos a formato BSON. Inicialmente se analizaron herramientas online como la de Safe Software [66] para realizar la migración de los datos. Ofrecen una versión de prueba gratuita de 30 días. Tras descargarla y empezar a usarla, debido a la complejidad de la aplicación y de los datos originales y sus relaciones, se optó finalmente por una migración manual.

Aunque algo más laboriosa, la migración manual permitió una mayor flexibilidad para almacenar los datos en base a nuestras necesidades. También nos sirvió para profundizar en el conocimiento estructural de la base de datos, así como de la fuente de información que se pretendía migrar. La forma de llevar a cabo dicha migración ha sido la siguiente:

1. Creamos una consulta SQL en el programa SQL Server 2014 Management Studio [67]. Según los datos de la tabla que queramos migrar y el volumen de ésta, será una consulta más o menos compleja. Cada campo que se vaya a devolver estará orientado a tener una estructura lo más próxima a un BSON.
2. Copiamos todos los registros obtenidos en nuestro editor de texto. Tenemos que tener en cuenta que cada registro será un futuro documento de una colección, y como tal debe tener una estructura en formato BSON.
3. Como es más que posible que haya campos con valores devueltos vacíos o nulos, debemos hacer un reemplazo masivo para limpiar cada conjunto de datos BSON.
4. Copiamos todos los datos procesados, y los trasladamos a su colección correspondiente que encontramos dentro de Robomongo.
5. Si todo es correcto, podremos ver la colección en Robomongo con todo el listado de colecciones nuevas. En caso de obtener algún error en la acción de salvado, deberemos corregir los datos procesados con anterioridad, puesto que el fallo se deberá a un error de sintaxis del formato BSON tratado (este podría ser, por ejemplo, una coma perdida, un mal cierre de una llave, o el olvido de unas comillas dobles). Una vez solventado, deberemos corregirlo en la consulta SQL que montamos en el paso 1, y volver a seguir los pasos hasta llegar a un buen funcionamiento del paso 5.

Todo este proceso se ha repetido para cada una de las colecciones que hay actualmente en nuestra base de datos de MongoDB.

5.1.2 Personalización del editor

Para el desarrollo del proyecto se ha optado por Visual Studio Code, un potente editor con amplias funcionalidades para trabajar también con Git y con ventana de consola propia.

Una de las principales ventajas de Visual Studio Code es que puede ser personalizado para adaptarlo al estilo necesidades de cualquier desarrollador. Para

este proyecto se han buscado los complementos más útiles, reduciendo en la práctica el uso a los siguientes:

- **Emmet.** Es un complemento esencial que ahorra mucho tiempo a la hora de escribir código, ya que permite utilizar abreviaturas para las etiquetas o crear clases en documentos CSS y HTML.
- **TSLint y ESLint.** Ejecutar un revisor suele ser más rápido que ejecutar la aplicación o las pruebas de unidad, por lo que es una buena idea ejecutar este complemento todo el tiempo para que esté continuamente controlando y revisando el código. Así se asegura un estilo uniforme y se mejoran las prácticas de programación.
- **Preattier.** Reorganiza y sangra el código para ser más legible.

5.1.3 Instalación de paquetes y librerías

Estos son los paquetes y librerías que se han ido instalando a lo largo de todo el proyecto según se ha ido necesitando las funcionalidades de la aplicación:

5.1.3.1 Babel

Babel.js es un transcompilador que convierte nuestro código ES6 en código de ES5, ya que no todos los navegadores, o al menos en su totalidad, son capaces de interpretar perfectamente ES6. La solución es pasar nuestros JavaScripts ES6 a ES5 [38] [39] [40].

Se utiliza de modo conjunto a Node.JS, y con él funcionando podríamos operar con React.JS de una forma más cómoda.

```
npm install --save-dev babel-cli babel-preset-env babel-preset-stage-3
```

5.1.3.2 Nodemon

Es un paquete que te permite actualizar tu web automáticamente en el navegador tras realizar algún cambio en el código, y así no tener que estar recargando la página constantemente.

```
npm i nodemon –S
```

5.1.3.3 Express

Es un framework de Node.JS que nos permite hacer más fácilmente conexiones en nuestra aplicación, configurar el entorno y tareas comunes en desarrollo y publicación web.

```
npm i express -S
```

5.1.3.4 Body-Parser

Es una librería que nos facilita tratar los datos que recibamos desde el cliente a través de las peticiones HTTP (POST, PUT, etc.).

```
npm i body-parser -S
```

5.1.3.5 Apollo-server-express

Es una librería que nos permite integrar Express con GraphQL por medio de Apollo.

```
npm i apollo-server-express -S
```

5.1.3.6 GraphQL

Herramienta desarrollada por Facebook donde facilita el crear una API para usar RESTful recibiendo y mandando sólo y exclusivamente la información que el usuario quiere.

```
npm i graphql -S
```

5.1.3.7 GraphQL-tools

Es un paquete que permite construir esquemas de GraphQL y resolverlos en JavaScript [41].

```
npm i graphql-tools -S
```

5.1.3.8 Mongoose

Es un paquete que nos permite integrar MongoDB con NodeJS para poder manejar más cómodamente los datos de nuestra base de datos no relacional con nuestra aplicación.

```
npm i mongoose -S
```

5.1.3.9 Merge GraphQL Schemas

Es un módulo que nos permite unificar todos los schemas y resolvers que tengamos organizados en archivos distintos para GraphQL, y los unifique luego en uno solo [42].

```
npm i merge-graphql-schemas -S
npm i path -S
```

5.1.3.10 Create React App

Es una utilidad para conectar el lado cliente con GraphQL, pero necesita de npm v5.2 en adelante, pues con la 5.1 o menos dio problemas en nuestra aplicación, por lo que tuve que actualizarlo, obteniendo la 6.1.0 [43].

```
npm update -g npm
```

5.1.3.11 React Router

Es una herramienta que nos permite crear los enrutadores para React, y de este modo será más fácil la navegación entre páginas [44].

```
npm i react-router-dom -S
```

5.1.3.12 Heroku

Es un servidor externo donde colgar la aplicación, la cual será subida mediante comandos Git.

Tras subirlo todo, nos dará una url donde probar que todo es correcto, y si falla, lanzar "*heroku logs --tail*" para ver por consola el posible error (como por ejemplo que no se puede conectar a la base de datos de Mongo).

5.1.3.13 MLab

Al necesitar de MongoDB en nuestra aplicación, Heroku tiene una utilidad para conectarse a una base de datos MongoDB mediante una conexión con MLab, la cual se encargaría de dar el alojamiento a nuestros datos. El problema es que, a pesar de ser una herramienta gratuita que ofrece Heroku, la configuración de esta utilidad desde dentro de ella, requiere el guardar los datos de una tarjeta de crédito del usuario que se da de alta como posibilidad para la compra otros servicios más.

Para evitar esto, tan sólo tenemos que ir directamente a la propia página de mLab [45], y tras crear una nueva cuenta de usuario en su web, creamos la base de datos, el usuario y password al que se conectará, y dentro de Heroku establecemos las variables de entorno proporcionadas por mLab para que nuestra aplicación pueda conectarse sin problemas a la base de datos [46] [47].

```
MONGOLAB_URI: mongodb://aaznar:sopde$2018@ ds143971.mlab.com:43971/diputacion_turismo
MONGODB_URI: mongodb://aaznar:sopde$2018@ ds143971.mlab.com:43971/diputacion_turismo
```

Para hacer la subida de BBDD a producción:

Exportar

```
mongodump -h ds143971.mlab.com:43971 -d diputacion_turismo –u <user> -p <pass> -o <directory>
```

Importar:

```
mongorestore -h ds143971.mlab.com:43971 -d diputacion_turismo -u <user> -p <pass> <directory>
```

5.1.3.14 Semantic UI-React

Es un framework que permite hacer más fácil y dar mucha más funcionalidad a la maquetación por estilos de los componentes que creemos en React [48].

```
npm i semantic-ui-react -S
npm i semantic-ui-css -S
```

5.1.3.15 Mongoose validator

Es una librería que permite validar nuestros campos de mongoose con un sin fin de utilidades de validación (email, monedas, longitud de campos, etc.) [49].

```
npm i mongoose-validator -S
```

5.1.3.16 React Truncate HTML

Es una librería que permite recortar el texto plano o HTML para mostrarlo por pantalla, de forma que el usuario ver una primera vista previa de la información que puede obtener de forma completa profundizando en la navegación [50].

```
npm i react-truncate-html -S
```

5.1.3.17 React Lazy Load Image Component

Es una librería que permite ir cargando las imágenes conforme se necesiten en la web, es decir, que conforme esa imagen se vaya mostrando en la página, irá siendo solicitada al servidor para que sea mostrada [51].

Con esto evitamos se consuma datos de forma innecesaria. Supongamos que por ejemplo tenemos un listado de 10 productos, cada uno con su foto correspondiente y 2MB de peso cada uno. Serían un total de 20MB la carga de esa página (contando sólo con el grueso de las fotos). Si no optimizáramos la web, el visitar esa página nos costaría un mínimo de 20MB, pues se descargan todas. Sin embargo, con la optimización aplicada actualmente, el usuario iría descargando poco a poco según esté por pantalla el que se tenga que mostrar la foto. Si está “oculta” por situarse en un scroll más abajo, no se llega a descargar nada. Con lo cual, nos ahorramos de forma sustancial consumo de ancho de banda.

```
npm i react-lazy-load-image-component -S
```

5.1.3.18 React Slick

Es una librería que permite hacer una galería dinámica de imágenes con diversos estilos y efectos, de forma que estas vayan pasando de forma automática a modo de diapositivas [53].

```
npm i react-slick -S
npm install slick-carousel
```

5.1.3.19 React Map GL

Es una librería que permite pintar mapas de opensource (OpenStreetMap) [53], evitando de esta forma el tener que recurrir a GoogleMaps, cuyas limitaciones de uso diario se ha visto reducido de forma drástica en los últimos meses, primando el pago por sus servicios.

Para poder utilizar, necesitaremos registrarnos previamente en MapBox, de donde nos será asignado un token, el cual podremos usar en la llamada externa para el pintado de nuestros mapas [54]

```
npm i react-map-gl -S
import 'mapbox-gl/dist/mapbox-gl.css';
```

5.1.3.20 React Audio Player

Es una librería que permite pintar un reproductor de audio para poder escuchar audios online desde nuestra propia web [55].

```
npm i react-audio-player -S
```

5.1.3.21 React Player

Es una librería que permite pintar un reproductor de video para poder ver online los vídeos enlazados a YouTube o Vimeo [56].

```
npm i react-player -S
```

5.1.3.22 React share

Es una librería que permite incluir botones de redes sociales para compartir la página a las redes del usuario [57].

```
npm i react-share –S
```

5.2 Estructura de la aplicación

5.2.1 Directorios

El código desarrollado en la aplicación se organiza en la siguiente estructura de directorios:

- **información-turistica-client.** Código que se ejecuta en el lado cliente.
- **config.** Contiene los ficheros de configuración de la aplicación.
- **models.** Definimos todos los modelos de datos.
- **resolvers.** Realizamos las acciones para obtener o manipular datos.
- **schema.** Definen cada uno de los resolvers que utilizará cada modelo de datos.
- **types.** Almacena la estructura tipificada de cada modelo de datos para que pueda ser usado con GraphQL.
- **utils.** Contiene varios ficheros para facilitar la programación.
- **node_modules.** Contiene todas las librerías y paquetes instalados en la aplicación
- **extra.** Es una zona de almacenamiento que será incluida con la finalidad de tener copia de respaldo, aunque directamente no son usados en nuestra aplicación.

En cada uno de los siguientes apartados se detalla su estructura.

información-turistica-client

Contiene todos los ficheros que se ejecutan en el lado cliente (estilos, vistas de componentes, etc.):

- **build.** Directorio que se despliega para ser ejecutado en producción.

- **node_modules.** Alberga todas las librerías y paquetes instalados para esta versión de cliente
- **public.** Almacena la información pública, como por ejemplo el icono que aparecerá en la dirección del navegador (“favicon.ico”)
- **src.** Es el directorio donde encontraremos el grueso de lo que visualmente ve el usuario. Está dividido en varias carpetas:
    - **components:** Contiene los distintos componentes usados en la aplicación (zona de búsqueda, listado de artículos, ficha de un artículo, …)
    - **css:** Donde se guardan los distintos estilos CSS que maquetan la web.
    - **routes:** Establece la organización del sistema de navegación de la web, fijando el encaminamiento de cada una de las zonas de la web.
    - **utils:** Además de contener funciones comunes para facilitar la programación y hacer un código más limpio, posee los ficheros “*queries.js*” y “*query.js*” que se encargarán de guardar las llamadas a GraphQL que se harán internamente en la aplicación para obtener respectivamente listados de productos según una determinada categorización, o la información de un producto en concreto.

config

Aquí se guardan los ficheros de configuración de la aplicación, como por ejemplo las variables de entorno, las cuales dotaremos de dos posibles valores. De esta forma, según se esté en desarrollo o producción, se usará uno de los valores u otro. Esto es importante por ejemplo para conectarnos a la base de datos.

```
module.exports = {
    port: process.env.PORT || 5000,
    app_name: process.env.APP_NAME || 'informacion-turistica',
    graphiql: process.env.GRAPHIQL || 'true',
    db: process.env.MONGODB_URI || 'mongodb://localhost:27017/diputacion_turismo',
    mongodb_user: process.env.MONGODB_USER || 'root',
    mongodb_password: process.env.MONGODB_PASSWORD || 'password',
    mongodb_debug: process.env.MONGODB_DEBUG || 'true',
    token_map: process.env.TOKENMAP || 'pk.eyJ1IjoiYWF6bmFyIiwiYSI6ImNqa3V5bzR4NTBlYWMzcWxsaTBxOTM2dzQifQ.X3rhPUNVtYT2TMFaOnPQBg',
    SECRET_TOKEN: 'miclavedetokens',
  }
//mongodb://aaznar:sopde$2018@ds113442.mlab.com:13442/diputacion_turismo
```

Figura 6 - Variables de configuración

Como vimos en el anterior apartado (**5.1.3.13 Instalación de paquetes y librerías: mLab**), en Heroku definiremos las variables de entorno que nos permitirán conectarnos a nuestra base de datos de MongoDB de producción, mientras que en el entorno de desarrollo tendremos otra absolutamente distinta (línea 5 de código de la *Figura 8*).

models

Aquí están definidos los esquemas que creamos de cada tipo de dato para poder asociaros a su correspondiente colección de MongoDB. Dichos esquemas se crean gracias a la librería “mongoose”.

resolvers

Aquí clasificaremos la forma de obtener o manipular los datos en tres carpetas:

- mutations: Contiene el desarrollo de funciones que son llamadas por los distintos tipos de modelos de datos para poder ser manipulados. Esto es, crear, modificar o eliminar datos.
- queries: Contiene el desarrollo de funciones se encargan de obtener los datos de forma genérica, bien sea para un artículo en concreto o para un conjunto de ellos de una determinada categoría.
- types: Contiene el desarrollo de las funciones que se encargan de obtener la información adicional de otras colecciones que están relacionadas con las colecciones principales de los modelos de datos a consultar.

schema

Contiene el listado de nombre de funciones que serán luego usadas en los resolvers descritos anteriormente, distinguiendo entre las que hacen manipulación de los datos (*mutations*) y las que sólo obtienen datos (*queries* para los modelos principales de la aplicación, y *types* para los modelos de datos relacionados con esos principales).

types

Alberga cada uno de los esquemas de datos que serán luego usados por GraphQL para ser consultados. Están fuertemente tipados, y sus estructuras deben ser como la de los modelos de datos, sólo que con la nomenclatura de GraphQL (por ejemplo, en ésta última se usa el tipo ***Int*** para declarar a una variable, mientras que para nuestro modelo de datos creado con mongoose se usa ***Number***).

utils

Encontraremos ficheros que nos permiten crear funciones que serán llamadas desde diversos puntos de nuestro proyecto, de forma que la programación sea más limpia y fácil de programar.

node_modules

Cualquier archivo situado en este directorio no se carga en ningún lugar. Los paquetes NodeJS instalados en los directorios *node_modules* deben importarse mediante la importación o mediante el uso de dependencias en package.js, que se encuentra en la raíz del proyecto.

extra

Este directorio es como un repositorio local del que podamos hacer también copias de respaldo, pues contiene información como los BSON de cada colección que conforma nuestra base de datos en MongoDB, o las consultas SQL que hicimos inicialmente para poder migrar los datos de SQL Server a MongoDB. Ha sido muy útil porque conforme hemos ido avanzando en el proyecto, se han podido detectar errores en las exportaciones de datos y con ellos ha sido más sencillo de volver a migrarlos ya corregidos.

5.2.2 Colecciones de MongoDB

En MongoDB los **documentos** se agrupan en **colecciones**. Estableciendo un paralelismo con las bases de datos tradicionales, los documentos vendrían a ser lo que los registros en el mundo relacional. Las colecciones son parecidas a las tablas del mundo relacional, aunque no imponen una estructura fija a los documentos que contienen, ni siquiera al tipo de datos de cada campo.

Podríamos decir que en MongoDB la estructura de los datos (bases de datos, colecciones, campos, …), es implícita, flexible y dinámica. Implícita porque lo normal es que los documentos de una colección compartan estructura, y ésta no se declara de manera explícita antes de introducir los documentos. Flexible porque esa estructura no se impone a ningún documento. Y dinámica porque la estructura puede cambiar.

Esto no quiere decir que cuando hagamos una aplicación no tengamos que realizar un proceso de análisis de la información a guardar en MongoDB. Al contrario, hay que hacerlo (y de hecho lo hemos hecho), aunque de manera diferente a cómo se hace en el mundo relacional (normalización). Por ejemplo, algunos de los aspectos más importantes a la hora de diseñar nuestra estructura de datos en MongoDB son la forma en que se consultan los datos, la forma en que se particionan (en caso de necesitarlo) y los requisitos de atomicidad.

Para nuestro caso, como comentamos anteriormente, por medio de la librería Mongoose, creamos en el modelo de datos la estructura de cada colección y el tipo de información que se va a guardar en ellos. De esta forma, nuestra base de datos estará compuesta por una serie de colecciones principales, las cuales pueden tener asociadas otras colecciones que dotan de más características a los documentos de esas colecciones.

Debido al gran volumen de información, se ha optado por dividir ésta en cinco bloques (colecciones) principales. Dichos bloques son los siguientes:

- **artículos_establecimientos.** Almacena la información de alojamientos, restaurantes, bares... de la provincia de Málaga.
- **artículos_informacion_ocio.** Alberga la información de oficinas de turismo, medios de transporte, campos de golf… En definitiva, cualquier empresa o servicio de información y ocio para el usuario, dentro de la provincia de Málaga.
- **artículos_naturaleza.** Dispone de una amplia información de zonas naturales de interés, como por ejemplo playas, parques naturales o cuevas; o de distintas especies de flora y fauna que se dan en la provincia.

- **artículos_patrimonio.** Almacena gran cantidad de lugares culturales a visitar en la provincia de Málaga (museos, monumentos, lugares emblemáticos, plazas, …).
- **artículos_tradiciones.** La historia de los municipios también tiene cabida en nuestra base de datos, así como leyendas famosas que perduran en la historia de la provincia, personajes famosos o fiestas populares de cada rincón de Málaga.

Dentro de cada una de estas colecciones hay asociados contenidos multimedia (imágenes, vídeos, audios y/o documentos), los cuales están almacenados en las siguientes colecciones:

- **audios.** Posee una amplia y variada documentación sonora relacionada con los documentos de las distintas colecciones principales.
- **documentos.** Guarda una gran cantidad de documentos relacionados con esas colecciones principales.
- **imágenes.** Amplísimo catálogo gráfico de los distintos documentos de las colecciones principales. Muchas de las imágenes referenciadas incluso han sido utilizadas en libros de información turística ofrecidos por la Excma. Diputación de Málaga [59].
- **videos.** Subidos en su mayoría al canal de YouTube de la Excma. Diputación de Málaga [60] [61] [62], aquí se tiene un amplio registro de cada uno de los vídeos que se han ido usando en la web de dicha administración.

Las siguientes colecciones que restan sirven para definir más características de ese bloque principal de colecciones o para categorizarlas:

- **artículos_tipos.** Contiene información que categoriza las distintas colecciones principales (si una colección es un ave, una cueva, un museo, etc.)
- **artículos subtipos.** Es un nivel más profundo de la categorización de las colecciones principales, dentro de los tipos de estos anteriormente vistos.
- **archivos_grupos.** Contiene distintos grupos que sirven para categorizar las colecciones de contenido multimedia.

- **documentos_subtipos.** Guarda diversos tipos de datos que categorizan a las colecciones multimedia de *documentos*.
- **localizaciones.** Contiene el posicionamiento geográfico de zonas de interés turístico, de forma que puedan ser utilizadas esas coordenadas para mostrarse a modo de mapa en la aplicación web.
- **lugares.** Sus *documentos* son todas las comarcas y municipios de la provincia de Málaga, de forma cada *documento* de las colecciones principales pueda ser situado específicamente en alguno de ellos.

En la *Figura 7* puede verse un sencillo esquema de cómo están relacionadas las colecciones entre sí.

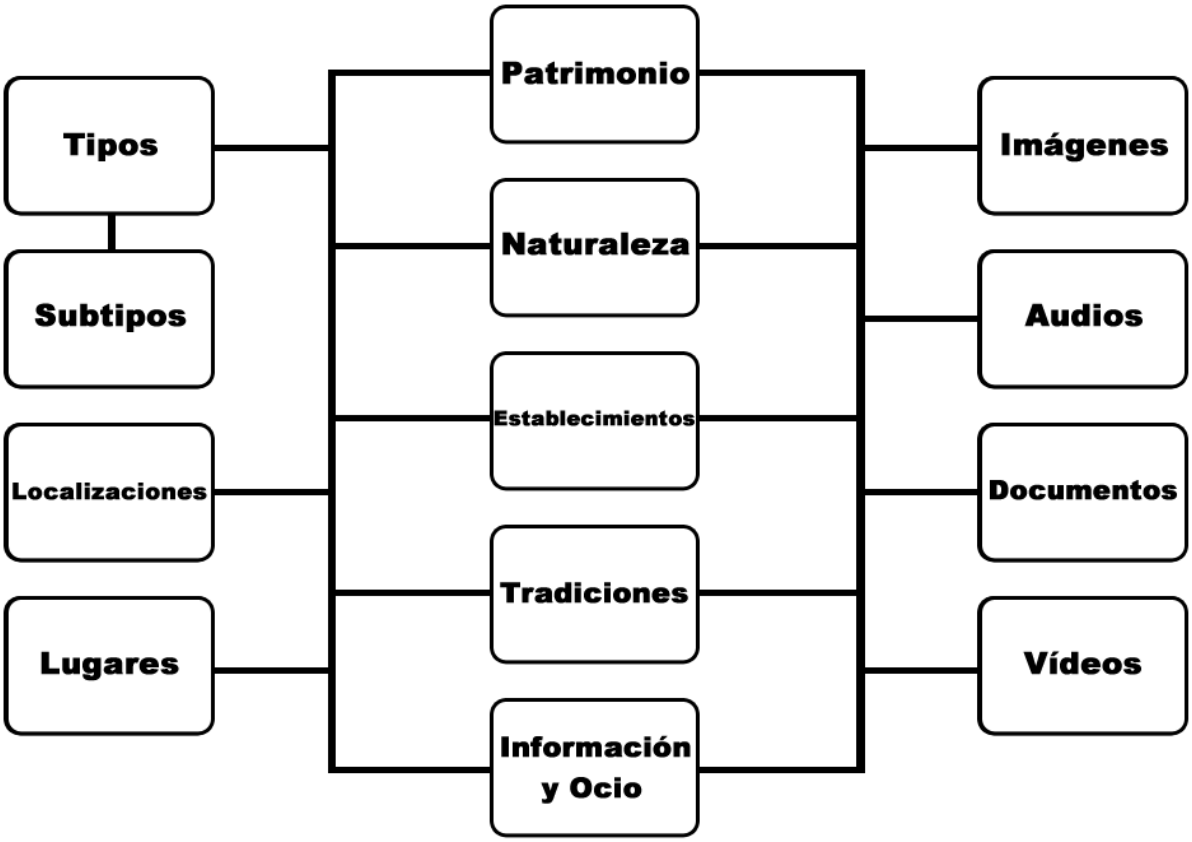


Figura 7 - Esquema básico de las colecciones en MongoDB

Además de todas estas colecciones, se han migrado de la base de datos original en SQL Server las siguientes colecciones, con el fin de ampliar la funcionalidad y la cobertura informativa de la aplicación web:

- **artículos_publicaciones.** Posee información tales como folletos, planes de estudios, normativas, …. En resumen, cualquier tipo de publicación de carácter pública desarrollada por la Excma. Diputación de Málaga.
- **artículos_provincia.** Información más detallada de cada municipio tiene cabida en esta colección, como por ejemplo los ayuntamientos de toda la provincia, donde están situados, descripción característica de cada municipio, población, etc.
- **canales.** Con la finalidad de poder en un futuro dotar de más flexibilidad y organización en la expansión del portal web turístico, cada documento de la colección, ya sea esta multimedia o principal, tiene un canal asignado. Dicho canal puede ser desde un Departamento de la Excma. Diputación de Málaga, un Servicio o un Área, a un municipio propio, por ejemplo. Con ello podemos identificar rápidamente si un *documento* será mostrado dentro de esa sección de la web o no. Supongamos que una imagen tiene los canales del Área de Deportes y del Servicio de Turismo. Ello quiere decir que, si en el portal web existiera por ejemplo una galería de imágenes dentro de la sección del Área de Deportes y del Servicio de Turismo, dicha imagen podría ser vista por el usuario. Sin embargo, si entrásemos por ejemplo en la Delegación de Medio Ambiente, al no estar su canal asignado, no se podría ver dentro de su galería de imágenes.
- **propietarios.** Esta colección contiene todos los posibles propietarios del portal web. En toda acción que se hiciera sobre los documentos de las colecciones principales o multimedia (crear un documento, modificarlo o eliminarlo), se guardaría quién lo ha llevado a cabo. También puede ser una forma de acotar ciertas acciones sobre determinados usuarios, a modo de permisos. Por ejemplo, sólo podrán modificar imágenes una serie de determinados propietarios.
- **canalizados.** Contiene una relación entre los propietarios que realizan la acción sobre los *documentos*, y a donde quieren que sea mostrado esa información. Por ejemplo, el propietario de la Delegación de Medio Ambiente podría estar interesado en que una imagen nueva obtenida sea visible en el Servicio de Turismo, por lo que le asignaría a su canal correspondiente, además de en el de su propia Delegación. De esta

forma se va extendiendo el que un *documento* pueda ser utilizado en diversas zonas del portal.

5.2.3 Esquemas de GraphQL

Como ya hemos visto, necesitaremos de dotar de una estructura a nuestros esquemas en GraphQL de igual forma que hacemos con los modelos de datos gracias a Mongoose. Ambas son parecidas en estructura y forma, salvo para aquellos modelos que están relacionados con las colecciones principales anteriormente vistas. Además de un cuerpo parecido, en GraphQL vamos a crear una capa más por encima para que nos ayude con la paginación de elementos que se muestren en la web [63] [64].

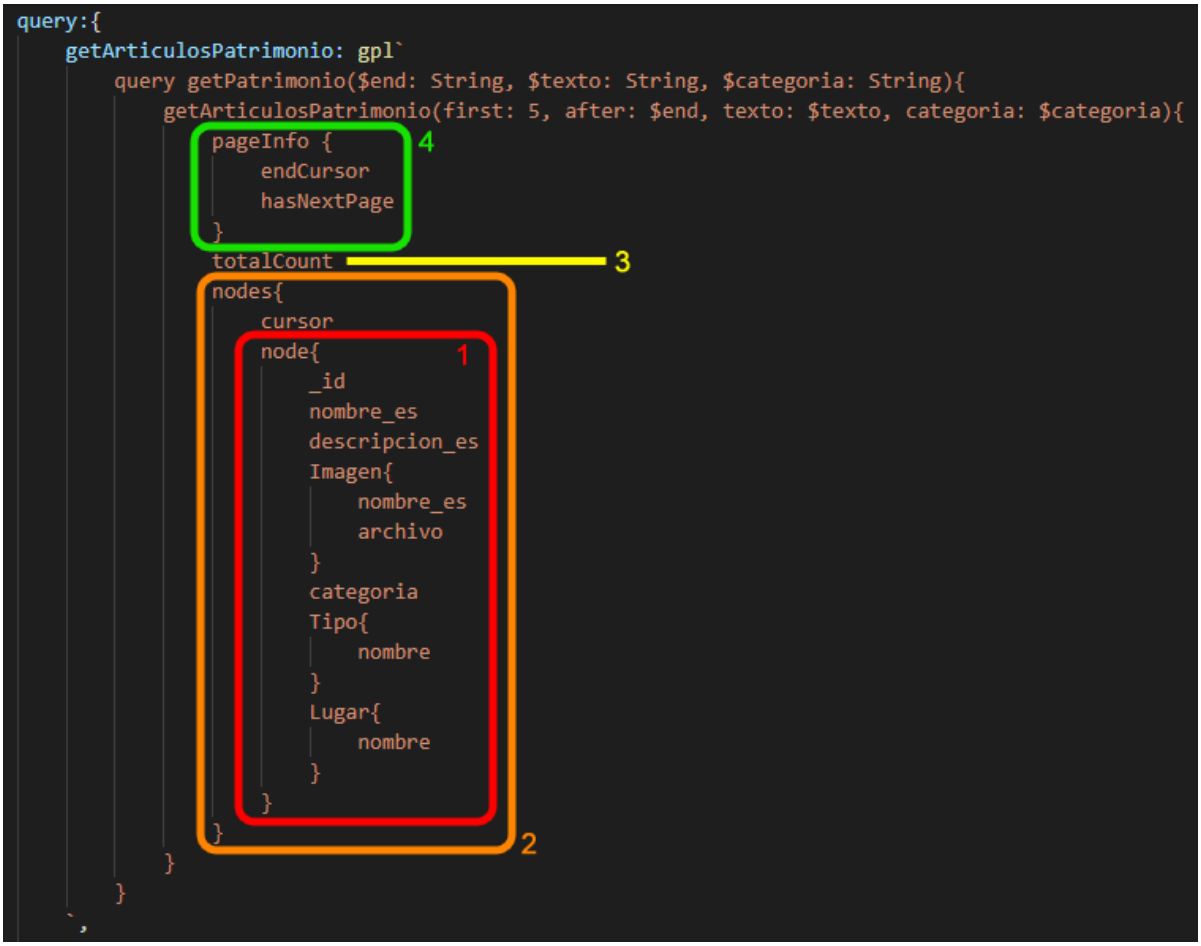


Figura 8 - Estructura de datos en GraphQL

Tal y como se puede observar en la *Figura 8*, hay varias zonas remarcadas por colores.

- **Zona 1.** Donde se devuelve la información del documento almacenada en nuestra base de datos de MongoDB.
- **Zona 2.** Capa que engloba toda la información a mostrar visualmente para una mejor organización de los datos. Además, mediante la propiedad ***cursor*** mostraría la posición actual que tendría ese *documento* a modo de puntero.
- **Zona 3.** Número de elementos totales que tiene realidad la consulta, puesto que a priori se mostrará solo una pequeña parte de ellos, según la configuración de paginación establecida.
- **Zona 4.** Contiene la información de la página actual, mostrando en la propiedad ***endCursor*** el puntero del último elemento que se muestra en dicha lista de *documentos*; y si hay más datos que mostrar en la propiedad ***hasNextPage***, para informar al programador si ha llegado o no al final de los datos totales.

5.3 Interfaz

Una vez encajadas todas las piezas de nuestro proyecto, donde la programación del lado servidor está conectado con el lado cliente, y se permite una interacción para la solicitud de los datos, procedemos a montar la parte visual de la aplicación: la web pública que podrá ser visitada por cualquier usuario con conexión a internet.

De forma análoga a como se describen las colecciones principales de nuestra base de datos en MongoDB, la parte web constará de 5 pilares fundamentales:

- Patrimonio
- Naturaleza
- Tradiciones
- Establecimientos
- Información y Ocio

5.3.1 Funcionalidad

Por medio de un sencillo menú superior, podremos visitar cualquiera de las secciones descritas, y según la seleccionada, consultar su información correspondiente almacenada en nuestra base de datos. El proceso es sencillo:

1. El usuario selecciona una sección del menú.
2. Internamente por GraphQL se solicita la información de esa sección, pero sólo aquellos datos que necesita inicialmente: nombre, breve descripción, tipo de *documento* que es, su situación geográfica en la provincia, y una imagen que lo caracterice.
3. El lado servidor le devuelve la información en formato JSON.
4. El lado cliente pinta esa información a modo de listado.

Partiendo de ese listado inicial, el usuario podrá hacer uso de un filtro de búsqueda para personalizar más dichas búsquedas. Una vez encontrado el *documento* deseado, tan solo tiene que seleccionarlo y, sin recargar la página, se repetirá el proceso anteriormente descrito, solo que en vez de pedir por GraphQL cierta información genérica y básica de un listado de documentos, pediremos toda la información existente de ese *documento* seleccionado; y una vez obtenida, será pintada por pantalla.

Con estos simples pasos se demuestra visualmente lo importante y característico de la programación reactiva. Sin necesidad de recargas en la página podemos ver tanto un listado como una ficha.

A esta reactividad hay que sumarle una nueva funcionalidad más, y es la del "*scroll infinito*" [65]. La idea de esta funcionalidad es permitir al usuario navegar por tantos *documentos* como quiera y existan en esa *colección*. Inicialmente el usuario empieza con un listado de 5 o 10 artículos. Tal y como explicábamos en la instalación del paquete **React Lazy Load Image Component**, conforme tenga que verse la imagen de un artículo se irán solicitando al servidor y seguidamente cargándose para ser mostradas. A medida que el usuario siga bajando por el listado, se irá repitiendo el proceso de descarga de imágenes. Pero ¿qué pasa cuando el usuario llega hasta el final de la pantalla, en ese tope de 5 o 10 artículos? Pues que internamente, de forma reactiva, se solicita mediante GraphQL el siguiente grupo artículos a ser mostrados, y así constantemente según se llegue al

final de la página mediante scroll. Estas peticiones pararán de forma manual por el usuario por decisión propia, o bien porque ya no hay más elementos que mostrar. Esto se consigue gracias a la estructura de los esquemas de GraphQL ya vistos con anterioridad.

Si la propiedad devuelta por GraphQL en el JSON es *false*, entonces no permitirá mostrar más elementos. Cada vez que se muestra un listado de artículos se guarda en propiedad ***pageInfo.endCursor*** la posición del último de ellos (el valor que hay en la propiedad ***nodes.cursor*** del último de los elementos devueltos), y si se requiere de más datos tras hacer el scroll mencionado, se obtendría la siguiente tanda de artículos partiendo desde esa posición. Automáticamente ese "puntero" se actualizaría apuntando ahora al último elemento calculado, teniendo ya así nuevamente la posición actual desde la que volver a contar si fuera preciso. Como decíamos antes, las peticiones pararán por decisión personal del usuario al dejar de hacer scroll en la página, o bien porque se ha llegado al final del listado, lo cual se detecta porque internamente la propiedad ***pageInfo.hasNextPage*** viene devuelta en el JSON de GraphQL con un valor *false*.

5.3.2 Componentes

Para pintar la web y hacer una optimización de código limpio y sin redundancias, hacemos uso de los componentes creados con React. Para nuestro proyecto podemos distinguir dos pantallas principales: listado de elementos, y ficha de un elemento. En la pantalla del listado de elementos podemos encontrar los siguientes componentes:

- **Menú superior.** Se pintará en todas las páginas que visitemos, por lo que es necesario incluirlo como un componente que se pueda usar de forma dinámica a lo largo de la web. Su finalidad es sencilla: pintar los principales pilares de nuestra aplicación para que puedan ser seleccionados indistintamente, y mostrar su información correspondiente.
- **Listado de elementos.** Ya se seleccione el apartado que se quiera, todos usarán el mismo componente para mostrar la lista de documentos de dicho apartado. El poder de este componente está en que se le pasa internamente una serie de parámetros que permiten distinguir si usar una

consulta en GraphQL u otra, de forma que se pinte luego los datos recibidos en el JSON de la petición.

- **Filtro de búsqueda.** Según los parámetros que se le pase a este componente, se pintarán unos criterios de búsqueda u otros. En base a los criterios filtrados por el usuario, se irá devolviendo el correspondiente listado de artículos que coincidan con la búsqueda establecida.

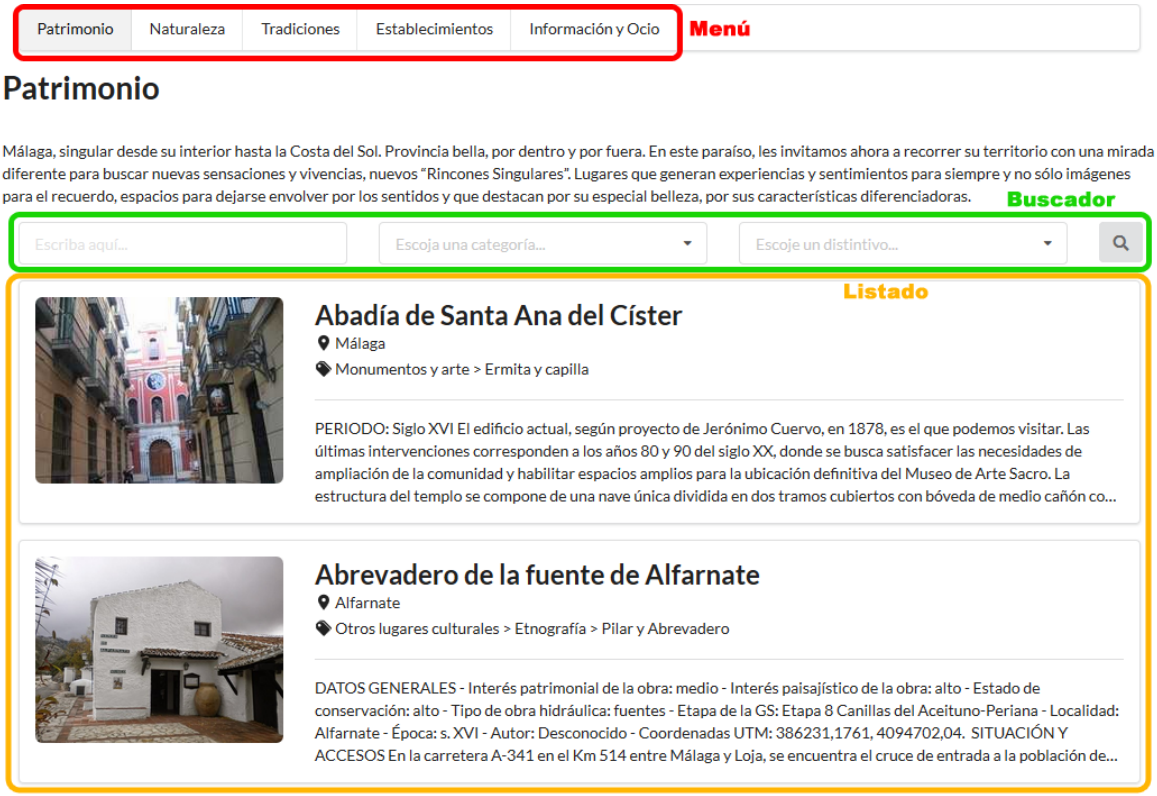


Figura 9 - Componentes usados en listado de datos

Tal y como se puede apreciar en la *Figura 9*, en la parte superior está el componente del menú en rojo, en verde el componente que dibuja los criterios de búsqueda, y en amarillo el componente que donde se listarán todos los elementos.

Para la pantalla de la ficha de un elemento determinado podemos encontrar los siguientes componentes:

- **Ficha de un elemento.** Se accede a este componente siempre que se seleccione un artículo del listado. En dicho listado aparece una información básica y destacable de cada artículo, pero una vez dentro de su ficha, podremos ver una información más detallada del artículo

seleccionado, como por ejemplo localización geográfica en un mapa, galería de imágenes, vídeos y/o audios asociados, etc.

- **Mapa.** Componente que se encarga de pintar en un mapa de código abierto la localización exacta de un artículo pasado previamente por parámetro.
- **Slider.** Este componente se caracteriza por mostrar un carrusel automático de fotografías, a modo de diapositivas que van avanzando solas. Dichas fotografías pertenecerán al artículo que se está visitando en ese momento. De igual modo que ocurre en el listado principal, conforme se van deslizando las imágenes de forma automática, se van solicitando al servidor y descargándose para ser mostradas, optimizando así el consumo de datos para beneficio del usuario.

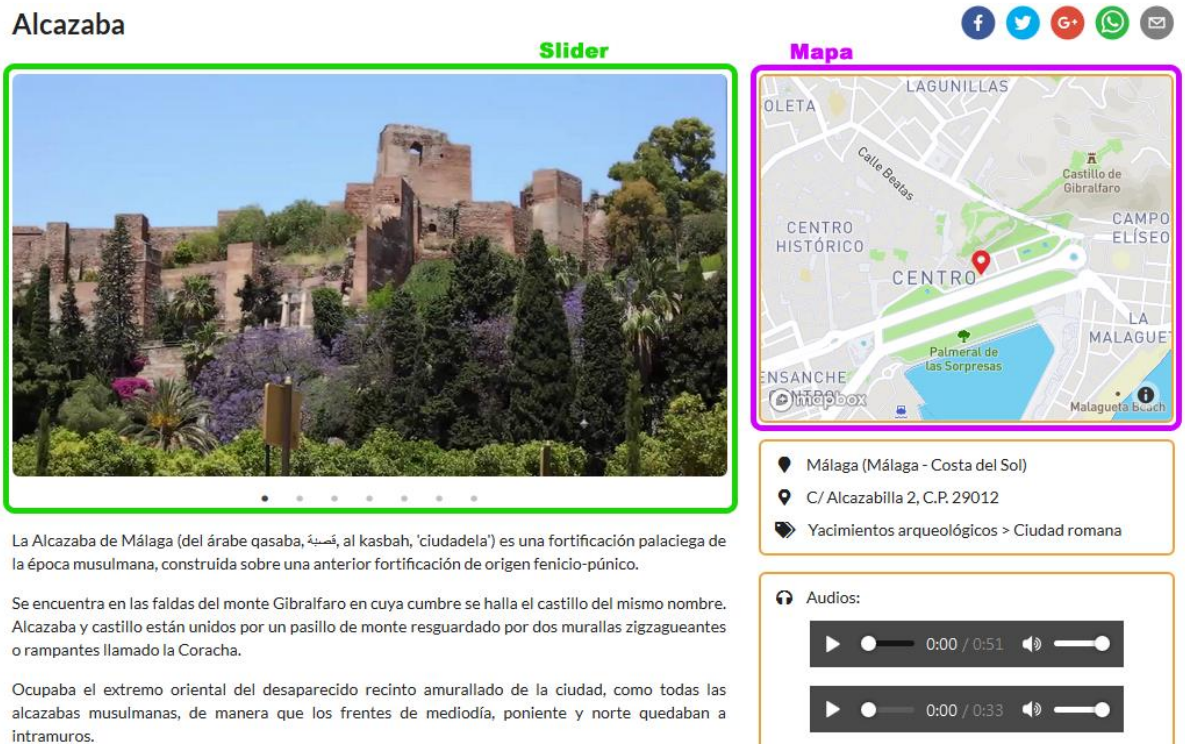


Figura 10 - Componentes usados en ficha de datos

En la *Figura 10* podemos ver algunos ejemplos de estos componentes (en verde el componente del visor dinámico de imágenes, y en morado el del mapa que localiza geográficamente al elemento), siendo la ficha en sí un propio componente más.

6. Despliegue

6.1 Entorno de desarrollo

Si queremos instalar el proyecto en nuestra máquina local y poder seguir desarrollando funcionalidades, deberemos tener instalado MongoDB en nuestro ordenador y bajarnos una copia de nuestro repositorio de Bitbucket.

```
https://andynedine@bitbucket.org/andynedine/informacion_turistica.git
```

Para ello deberemos usar Git en nuestro editor Visual Studio Code [84] para clonarlo. Antes de empezar la clonación nos pedirá un directorio de nuestro equipo donde guardarlo, para que seguidamente se pueda descargar todo el proyecto. Una vez finalizada dicha descarga deberemos proceder a la instalación de los paquetes del proyecto. Seguiremos los siguientes pasos:

```
cd informacion_turistica
npm install
cd informacion-turistica-client
npm install
```

Tras acabar la instalación de todos los paquetes deberemos lanzar el servicio de MongoDB para que podamos conectarnos al arrancar la aplicación, ejecutando simplemente el comando *mongod* desde la consola de comandos:

Para ejecutar tanto la aplicación como el cliente instalado dentro deberemos lanzar también desde consolas diferentes las siguientes instrucciones:

```
$<ruta_app>: npm start
```

```
$<ruta_app> / informacion-turistica-client : npm start
```

Una vez arrancado todo, ya podremos visualizar en el navegador de nuestra máquina local la aplicación web:

```
http://localhost:3000
```

6.2 Entorno de producción

En este apartado se explicarán los pasos que se han llevado a cabo para configurar el entorno de producción, la base de datos y el posterior despliegue de la aplicación.

6.2.1 Subida de la base de datos a entorno de producción

Debemos subir nuestra base de datos local en MongoDB al entorno de producción. Para ello, primero debemos hacer una exportación escribiendo en la consola de nuestro ordenador la siguiente instrucción:

```
mongodump -h localhost -d diputacion_turismo -o bbdd
```

Lo que hace esta instrucción *mongodump* de Mongo es ir al servidor local *localhost*, y de ahí coger la base de datos *diputación_turismo* y almacenar toda la información en un directorio local "*bbdd*".

El siguiente paso será acceder a la página de mLab, donde previamente deberíamos haber creado una cuenta de usuario. En caso de tener ya una base de datos *diputacion_turismo*, podemos sólo eliminar las colecciones. Si aún no tenemos nada de información guardada deberemos seguir los siguientes pasos:

1. Crear una base de datos con el mismo nombre *diputación_turismo* que tenemos en local.
2. Crear un usuario que nos permita escribir y leer los datos. En nuestro caso será el nombre de usuario "*aaznar*", y con la clave "*sopde$2018*".
3. Para importar los datos antes exportados de local, debemos ejecutar la siguiente instrucción en la consola de nuestro ordenador:

```
mongorestore -h ds143971.mlab.com:43971 -d diputacion_turismo -u aaznar -p sopde$2018
bbdd/diputacion_turismo
```

Lo que hace la instrucción *mongorestore* es ir a la ruta ofrecida por mLab en sus servidores *ds143971.mlab.com:43971*, y dentro de la base de datos *diputacion_turismo* creada previamente, con el usuario y password definidos y con permisos suficientes, se importará la base de datos localizada en el directorio de nuestra máquina local "*bbdd/diputacion_turismo*".

Una vez finalizado todo el proceso de carga, ya tendremos en producción nuestra base de datos.

Es importante el tener en cuenta que, si se ha hecho por primera vez la exportación, debemos especificar la ruta del servidor donde está alojada nuestra base de datos en las variables de entorno de nuestra aplicación. Por defecto, en código tenemos preparado que, según sea el entorno, use unos valores u otros. En local no hay problema porque están puestos directamente, pero para el entorno de producción deberemos crear esas variables y sus valores correspondientes, tal y como veremos en el siguiente punto.

6.2.2 Especificaciones del entorno de producción

Para el despliegue de la aplicación se opta por la plataforma Heroku, un servicio en la nube que permite alojar aplicaciones de carácter general desarrolladas con múltiples tecnologías.

El plan más básico es gratuito, pero tiene algunas limitaciones, como la parada del servicio tras media hora de inactividad que puede generar unos segundos de pausa en la aplicación hasta su reactivación.

Hasta que el proyecto esté abierto a los usuarios, esto no presenta un problema, verificando además la cuenta con una tarjeta de crédito para obtener un *dyno* con 1000 horas de ejecución al mes.

6.2.3 Configuración del entorno de producción

Una vez dado de alta en Heroku, es necesaria la creación de una aplicación en el servicio antes de hacer el despliegue. Esto se puede hacer de forma online dentro del propio Heroku siguiendo unos sencillos pasos que nos va describiendo, o bien hacerlo mediante instrucciones de consola. Ya sea de una forma u otra, necesitaremos descargarnos "Heroku CLI" [82], para que así podamos hacer o bien el alta de la aplicación por consola, o también la necesaria subida desde nuestro

repositorio aplicando *push*. Dicho esto, para alojar una aplicación a Heroku es necesaria la utilización de un repositorio local Git, el cual usamos desde el inicio para el control de versiones y mediante el cual haremos el despliegue.

Pero antes, debemos preparar nuestro código para producción. Al poseer un servidor local instalado dentro de nuestra aplicación, debemos desplegarlo para que sea interpretado en el entorno de producción. Para hacerlo debemos meternos desde la consola en el directorio donde está instalado, y desplegarlo con la siguiente instrucción:

```
cd información-turistica-client
npm run build
```

Esto provocará que se cree dentro de esa ruta una nueva carpeta ***build*** con los archivos que serán interpretados en el entorno de producción.

Una vez que está todo preparado en nuestro código, lo primero que tenemos que hacer es, si no lo habíamos hecho aún, conectarnos a Heroku por consola, de forma que todas las siguientes acciones que hagamos con comandos de Heroku se realicen sobre nuestra cuenta:

```
heroku login
```

Tras introducir nuestros datos, ya podremos subir nuestro código que tenemos en el repositorio local a Heroku. Previamente hacemos *commit* por si se nos ha olvidado de actualizar nuestro repositorio.

```
git init
git add .
git commit -m “preparando código para heroku”
```

Si aún no hemos creado nuestra aplicación en Heroku y lo queremos hacer por terminal, deberemos lanzar la siguiente instrucción:

```
heroku create <app_name>
```

Una vez creado, ya sea de forma online o manual por consola, nos situaremos en la aplicación para que podamos ir subiendo nuestro código a producción:

```
git remote –a <app_name>
```

Para confirmar que Heroku ha sido configurado como repositorio remoto, se puede usar el comando:

```
git remote -v
```

En el resultado se muestra tanto Heroku como Bitbucket, usado para el control de versiones remoto.

Ya sólo nos quedaría subir nuestro repositorio a producción. Para ello subimos la rama master, que es la principal. También se podría subir otra rama, pero habría que especificarlo en la instrucción. Las respectivas órdenes son las siguientes:

```
git push heroku master
git push heroku <rama>:master
```

Teniendo en cuenta que el nombre elegido para nuestra aplicación de Heroku es “información-turistica”, tras subirlo todo nos dará una url donde probar que todo es correcto (*https://informacion-turistica.herokuapp.com*), y si falla, deberemos lanzar la siguiente instrucción en consola para ver cuál es el fallo (como por ejemplo, que no se puede conectar a la base de datos de Mongo).

```
heroku logs --tail
```

6.2.4 Recomendaciones

El proceso anterior es válido para una sola instancia de aplicación. Si la aplicación crece con una base de usuarios más grande y el tráfico aumenta, obviamente será necesario escalar el sistema.

Aunque el plan básico es una opción válida para las primeras pruebas en producción, sus prestaciones no son suficientes para un servicio en producción abierto a los usuarios, por lo que es necesario un cambio de plan a la modalidad estándar.

La mayoría de las veces, la escala puede hacerse horizontalmente, lo que significa que se podrían agregar más instancias y con un balance de carga que encamine el tráfico a las instancias que tengan menos carga en cada momento. Con otras soluciones de escalado utilizando un clúster, por ejemplo, se podría escalar verticalmente añadiendo más núcleos al servidor.

7. Conclusiones y futuros trabajos

7.1 Conclusiones

7.1.1 Resultados

Los objetivos de este proyecto abarcan numerosos aspectos como realizar un estudio y valoración de las tecnologías de desarrollo web más actuales, demostrar la potencia de JavaScript como lenguaje único y poner en práctica la aplicación del paradigma de la programación reactiva y la arquitectura isomorfa en el desarrollo de una aplicación.

En la fase inicial de este proyecto se ha realizado un amplio estudio de las tecnologías web, analizando su evolución en los últimos años e incluyendo una valoración de los principales marcos de trabajo utilizados en el desarrollo con JavaScript. También se ha decidido por el uso de una base de datos no relacional frente a las tradicionales bases de datos relacionales.

En la segunda parte se ha realizado el desarrollo, el cual está dividido en tres grandes bloques:

- Programación del lado servidor, con ayuda de NodeJS y Express, para conectarnos a la base de daos en MongoDB.
- Programación del lado cliente, con el uso de React y la creación de componentes, de forma que el usuario pueda interactuar con la aplicación web de forma sencilla y fluida.
- Programación de la API GraphQL para que, mediante llamadas internas, éstas se conecten al lado servidor para devolver los valores deseados y puedan ser tratados en ya el lado cliente.

En resumen, desde el punto de vista del producto, se ha obtenido una aplicación totalmente funcional, usable, testada y lista para ponerla en marcha. Si bien se ha pecado de ser ambicioso al abarcar más de la cuenta antes de comenzar este proyecto y dotar de demasiadas funcionalidades a la aplicación, se ha cumplido el objetivo inicial, demostrando que es posible la creación de un portal web en JavaScript partiendo de conocimientos prácticamente nulos sobre las

tecnologías actuales del mercado, utilizando las ventajas del desarrollo isomorfo y reactivo.

7.1.2 Dificultades

Durante el desarrollo del proyecto se han encontrado y solucionado números problemas, algunos motivados por la falta de experiencia en tecnologías utilizadas, pero otros propios de estas tecnologías en sí mismas.

Una de las principales dificultades ha sido la elección del marco de trabajo, ya que en el mundo del desarrollo de aplicaciones actual hay un número enorme de opciones que, especialmente al principio, resulta abrumador. Hace unos años el mundo del desarrollo web era complejo, pero mucho más estable al estar formado por lenguajes tradicionales que aportaban soluciones bastante conocidas. Este panorama ha cambiado radicalmente a un escenario de lenguajes, marcos de trabajo y tecnologías que no existían hasta hace poco.

Por otro lado, usar tecnología muy nueva tiene un problema añadido, y es que no hay disponible tanta información como en arquitecturas más clásicas. También es cierto que, en compensación, hay una comunidad muy activa y entregada dispuesta a resolver dudas en los foros dedicados. De hecho, han sido varias las veces que, tras un fuerte bloqueo en el desarrollo del proyecto por falta de conocimientos suficientes en la nueva tecnología usada, he tenido que solicitar ayuda online en blogs, foros, proyectos de GitHub, tutoriales o video-tutoriales que he ido tomado como referencia para el correcto desarrollo de la aplicación [68] [69] [70] [71] [72] [73] [74] [75] [76] [77] [78].

Las tecnologías de desarrollo actuales no sólo son muy amplias, sino que también cambian rápidamente. Por un lado, continuamente aparecen versiones nuevas que no siempre son compatibles con las anteriores y, por otro lado, tecnologías o herramientas que son un estándar quedan obsoletas al ser superadas por otras nuevas en pocos meses.

También ha habido problemas de compatibilidad al utilizar software desarrollado por un elevado número de autores diferentes. Al instalar una biblioteca de terceros ha sido frecuente encontrar problemas de integración que fueron resueltos utilizando los foros de ayuda o consultando directamente con el autor.

Incluso la programación en sí ha sido un problema, en el sentido de que, viciado seguramente por el uso de tecnologías obsoletas donde la forma de darle funcionalidad a un problema se podía solucionar de sólo una o pocas formas más (pero todas semejantes en esencia), para estas nuevas tecnologías usadas se han encontrado cantidad de código distinto en el que, si bien la solución final era la misma, la forma de programarlo y llegar a ella no tenía nada que ver la una con la otra. Partiendo desde el uso de versiones distintas de TypeScript (algunas fuentes implementaban en ES5 y otras en ES6), a la manera de programar las clases o los componentes de React, usando métodos internos según el ciclo de vida de éstos [79], o haciendo uso de las variables de estado del componente, etc.

Otro problema importante ha sido la subida a producción del código. Mientras en entorno de desarrollo ha funcionado perfectamente en todo momento, el llevar el código a entorno de producción ha sido bastante dificultoso. Todo se debe a que al usar React, el manejo de las páginas son estáticas, y hace falta definirle perfectamente mediante *express* la ruta que deben seguir esas páginas en caso de ser llamadas. Además, ha habido que seguir detalladamente unas series de indicaciones que la librería *create-react-app* [83] nos define para poder completar con éxito la subida al entorno de producción. Hasta que no se consiguió realizar todos estos pasos a la perfección, no se pudo ver el proyecto hecho realidad de forma pública.

7.1.3 Valoración personal

A nivel tecnológico ha sido enormemente satisfactorio, ya que abarca numerosas disciplinas en la que se ha tenido que aplicar las aptitudes y conocimientos en materia de ingeniería del software, bases de datos, programación, metodologías de programación, etc.

El proyecto utiliza tecnologías muy nuevas, por lo que ha sido desde su inicio un reto marcado por la falta de conocimientos y experiencia en todas ellas, especialmente en arquitectura isomorfa o aplicaciones reactivas. Salvo conocimientos en metodología ágil (de la cuales utilizo Scrum en proyectos de mi actividad laboral), se ha partido de cero en todas las tecnologías, metodologías y herramientas utilizadas. Esto se ha planteado desde el principio como una enorme

oportunidad de aprendizaje en todos los ámbitos que abarca el desarrollo este proyecto.

La gran ventaja de partir desde cero en una nueva tecnología y método de trabajo es que en gran parte de la programación usada tiene que ser analizada y comprendida línea por línea. No basta con coger un poco de código de un lado, otro poco de otro, hacer ciertos retoques para que funcione, y listo. Me ha quedado completamente demostrado que es imposible usar código de terceros sin tener que saber exactamente cuál es la finalidad de cada una de esas líneas de código. Aunque nos faciliten la funcionalidad de nuestra aplicación, es indispensable saber el porqué de su uso, pues es 100% seguro que tendrá que ser reutilizado para añadir alguna acción extra más.

Debido a esto, la curvatura de aprendizaje ha sido realmente dura. Si bien ha sido un proceso lento y más largo de lo previsto, realmente esta es la forma en la que mejor se aprende: a base de solucionar un problema tras otro buscando y recabando información, e intentando ponerla en práctica.

Aunque el comienzo fue lento, siguiendo también video tutoriales para tener una visión general y cercana al mismo tiempo de lo que funcionalmente podría usarse en nuestra aplicación [80] [81], ha sido realmente provechoso ya no solo para el proyecto, sino también a nivel personal por adquirir conocimientos y estar algo más actualizado en lo que a nuevos marcos de trabajo se refiere.

. En lo personal, he encontrado motivador por tener que usar tecnologías completamente nuevas, y con el sólo hecho de conseguir haberle dado vida, con todo ese trabajo que tiene detrás, ya me siento más que orgulloso; y más teniendo en cuenta (por suerte) mi situación laboral y personal, donde el poder sacar tiempo es todo un preciado tesoro que no habría sido posible sin la ayuda de mi familia.

7.1.4 Líneas de Mejora

Aunque se han cumplido los objetivos marcados en un principio, por la limitación temporal y de recursos, no se han desarrollado todas las propuestas de nuevas funcionalidades que podrían implementarse de cara al futuro. Para finalizar, se sugieren a continuación algunos puntos de mejora a realizar en la aplicación

Ampliar a portal web

Aunque tenga un gran trabajo detrás, a priori una aplicación web debe tener un mayor atractivo visual para el visitante. Ésta debería dar un paso más para crecer y convertirse en un portal web donde nutrirlo de más información, como por ejemplo una página de inicio con algunos banners que enlacen a la zona más importante del portal, una zona de noticias actuales, o una cabecera dinámica donde mostrar impactantes imágenes de alguno de los artículos almacenados en nuestros sistemas, zona de contacto, etc.

Gestión de datos

En nuestro proyecto sólo hacemos uso de los datos para mostrarlos en la aplicación web. Sería indispensable crear una zona, a modo de Panel de Gestión de Datos, en el que previa autenticación, podamos entrar y por crear nuevos elementos, modificarlos o eliminarlos. Si bien se usan servicios GraphQL para obtener datos, también está preparado el proyecto con la programación de poder hacer llamadas que puedan manipularlos. Es decir, actualmente se podrían hacer llamadas adecuadamente en las que podríamos crear altas, modificaciones o eliminaciones de elementos. Sólo nos faltaría programar en el lado cliente el interfaz para poder manipular los datos, pues los servicios internamente están ya disponibles.

Del mismo modo, también podríamos ampliar estos datos incluyendo nuevas secciones como por ejemplo gestión de noticias, eventos o anuncios para ser también mostrados en el portal.

Aprovechamiento de todos los datos

Como ya mencionamos en el punto 6.2.2 *Colecciones de MongoDB*, se completó totalmente la migración de los datos en SQL Server a sus colecciones correspondientes, pero hay algunas de ellas que no son usadas actualmente en este proyecto. Con el fin de ampliar el portal web, es donde tendrían cabida esas colecciones, pues dotarían de sentido el poder usar información de forma controlada en una o varias zonas del portal enriqueciendo de contenido nuevas secciones de éste.

Diseño web adaptable

Actualmente, nuestra aplicación está diseñada para ser vista correctamente en un equipo sobremesa. Con el gran auge que hay hoy día con los teléfonos inteligentes y las tabletas, es más que interesante y necesario adaptar la web para que pueda ser vista también correctamente en el resto de plataformas posibles, cumpliendo así que sea una web adaptable (también conocida como *web responsive*). Esto se conseguiría rediseñando la web aplicando nuevos estilos en CSS3 para que detecte el ancho de pantalla del dispositivo del usuario, y según sea éste, aplicar unos estilos u otros para conseguir una correcta visualización de los datos.

Además de todo esto, también nos veríamos beneficiados en el posicionamiento de Google, pues penaliza a aquellas páginas que no cumplan con el que sean adaptables al resto de plataformas.

8. Referencias bibliográficas

[1] "¿Qué es NoSql?", [en línea], https://aws.amazon.com/es/nosql/

[2] "GraphQL vs. Rest", [en línea]. Junio 2017 [27 de junio de 2017], https://blog.apollographql.com/graphql-vs-rest-5d425123e34b

[3] "GraphQL is the better REST", [en línea], https://www.howtographql.com/basics/1-graphql-is-the-better-rest/

[4] "React", [en línea], https://reactjs.org/

[5] "GitHub", [en línea], https://github.com/

[6] "ReactJS vs Angular5 vs Vue.js—What to choose in 2018?", [en línea]. Marzo 2018, [16 de marzo de 2018], https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d

[7] "Definición de metodología", [en línea]. Agosto 2018, [21 de agosto de 2018], https://es.wikipedia.org/wiki/Metodolog%C3%ADa

[8] "Definición de metodología ágil", [en línea]. Julio 2018, [29 de julio de 2018], https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software

[9] "Lean project management". 2007, https://www.pmi.org/learning/library/lean-project-management-7364

[10] D. J. Anderson, Kanban, 7 de Abril, 2010 ed., B. H. Press, Ed., 2010.

[11] "Trello", [en línea], https://trello.com/

[12] D. Crockford, O. Media, Ed., Mayo 2008.

[13] S. Stefanov, JavaScript Patterns: Build Better Applications with Coding and Design Pattern, O. Media, Ed., Septiembre 2010.

[14] "Arquitectura REST", [en línea]. Diciembre 2014, [3 de diciembre de 2014], https://es.slideshare.net/HctorFuentePrez/arquitectura-rest-42310133

[15] "Servicios web RESTful con HTTP", [en línea]. Noviembre 2013, [12 de noviembre de 2013], http://www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-i-introduccion-y-bases-teoricas

[16] "¿Por qué deberíamos abandonar REST y empezar a usar GraphQL en nuestras APIs?", [en línea]. Diciembre 2016, marzo 2017, [15 de marzo de 2017], https://www.genbeta.com/desarrollo/por-que-deberiamos-abandonar-rest-y-empezar-a-usar-graphql-en-nuestras-apis

[17] “GraphQL”, [en línea], https://graphql.org/

[18] “GraphiQL”, [en línea], https://graphql.github.io/swapi-graphql/

[19] “MongoDB”, [en línea], https://www.mongodb.com/es

[20] K. Chodorow, MongoDB: The Definitive Guide: Powerful and Scalable Data Storage, O'Reilly Media, Mayo 2013.

[21] “Apache CouchDB”, [en línea], http://couchdb.apache.org

[22] “Apache Cassandra”, [en línea], http://cassandra.apache.org

[23] “Digital Ocean”, [en línea], https://www.digitalocean.com

[24] “Heroku”, [en línea], https://www.heroku.com/pricing

[25] “OpenShift”, [en línea], https://www.openshift.com/products/features/

[26] “NodeChef”, [en línea], https://www.nodechef.com/nodejs-hosting

[27] “Robomongo”, [en línea], https://robomongo.org/

[28] D. Herron, NodeJS Web Development, P. Publishing, Ed., Junio 2016.

[29] “Características de React”. Octubre 2016, [20 de octubre de 2016], https://desarrolloweb.com/articulos/caracteristicas-react.html

[30] “Apollo server”, [en línea], https://www.apollographql.com/docs/apollo-server/

[31] “Apollo for React”, [en línea], https://www.apollographql.com/docs/react/

[32] “Pencil Project”, [en línea], https://pencil.evolus.vn

[33] “Visual Code Studio”, [en línea], https://code.visualstudio.com/

[34] “Git”, [en línea], https://git-scm.com/

[35] “Firebug”, [en línea], http://getfirebug.com

[36] “Chrome DevTools”, [en línea], https://developer.chrome.com/devtools

[37] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, Agosto 2008.

[38] “What is Babel?”, [en línea], https://babeljs.io/docs/en/index.html

[39] “Introducción a Babel.js y JavaScript ES6”, [en línea]. Abril 2016, [8 de abril de 2016], https://www.arquitecturajava.com/introduccion-babel-js-JavaScript-es6/

[40] “Qué es y para qué sirve Babel”, [en línea]. 2016, https://platzi.com/blog/que-es-babel/

[41] “GraphQL tools”, [en línea], https://www.apollographql.com/docs/graphql-tools/

[42] “Merge GraphQL schemas”, [en línea]. 2017, septiembre 2018, [23 de septiembre de 2018], https://github.com/okgrow/merge-graphql-schemas

[43] “Create React App”, [en línea]. 2016, septiembre 2018, [21 de septiembre de 2018], https://github.com/facebook/create-react-app

[44] “React Router”, [en línea], https://reacttraining.com/react-router/web/example/no-match

[45] “MLab”, [en línea], https://mlab.com

[46] “Como desplegar tu app NodeJ.js en Producción con Heroku”, [en línea]. Abril 2017, [26 de abril de 2017], https://www.youtube.com/watch?v=mkn5n8Ws-HE

[47] “Using MongoDB on heroku without verifying your account”, [en línea]. Febrero 2018, [24 de febrero de 2018], https://medium.com/@naumanzafarchaudhry/using-mongodb-on-heroku-without-verifying-your-account-9053a8c42e3c

[48] “Semantic UI-React”, [en línea], http://react.semantic-ui.com/usage/

[49] “Mongoose validator”, [en línea]. 2015, julio 2018, [25 de julio de 2018], https://github.com/leepowellcouk/mongoose-validator

[50] “React truncate HTML”, [en línea]. 2016, marzo 2018, [14 de marzo de 2018], https://github.com/jariz/react-truncate-html

[51] “React lazy load image component”, [en línea]. Febrero 2018, septiembre 2018, [15 de septiembre de 2018], https://github.com/Aljullu/react-lazy-load-image-component

[52] “React slick”, [en línea]. 2014, julio 2018, [15 de julio de 2018], https://github.com/akiran/react-slick

[53] “React Map GL”, [en línea], https://uber.github.io/react-map-gl/#/Documentation/getting-started/get-started

[54] “MapBox”, [en línea], https://www.mapbox.com/help/how-access-tokens-work/

[55] “React audio player”, [en línea]. 2015, mayo 2018, [23 de mayo de 2018], https://github.com/justinmc/react-audio-player

[56] “React player”, [en línea]. 2017, septiembre 2018, [20 de septiembre de 2018], https://github.com/CookPete/react-player

[57] “React share”, [en línea]. 2015, Agosto 2018, [26 de Agosto de 2018], https://github.com/nygardk/react-share

[58] “Diputación de Málaga – Turismo”, [en línea]. 2010, http://www.malaga.es/turismo

[59] “Guía Las Aves de la Gran Senda de Málaga. Guía de Observación”, [en línea]. 2017, http://www.gransendademalaga.es/es/4893/guia-aves-gran-senda-malaga-guia-observacion

[60] “Canal de YouTube de la Excma. Diputación de Málaga”, [en línea]. Noviembre 2011, [21 de septiembre de 2018], https://www.youtube.com/user/diputacionmlg/videos

[61] “Canal de YouTube de la Excma. Diputación de Málaga – Turismo”, [en línea], Mayo 2010, [18 de septiembre de 2018], https://www.youtube.com/user/turismodiputacion1

[62] “Canal de YouTube de Gran Senda de Málaga”, [en línea]. Enero 2014, [25 de abril de 2018], https://www.youtube.com/user/gransendamalaga

[63] “Infinite Scrolling in React using Apollo and React-Virtualized—GraphQL Cursor Pagination”, [en línea]. Marzo 2017, [4 de marzo de 2017], https://medium.com/@gethylgeorge/infinite-scrolling-in-react-using-apollo-and-react-virtualized-graphql-cursor-pagination-bf80617a8a1a

[64] “GraphQL Pagination”, [en línea], https://graphql.org/learn/pagination/

[65] “React Apollo Client GraphQL Cursor Infinite Scroll Pagination with GitHub API”, [en línea]. Junio 2018, [23 de junio de 2018], https://medium.com/@alfianlosari/graphql-cursor-infinite-scroll-pagination-with-react-apollo-client-and-github-api-fafbc510b667

[66] “Convert Data from SQL Server to MongoDB”, [en línea], https://www.safe.com/convert/sql-server/mongodb/

[67] “How to install SQL Server 2014 Management Studio”, [en línea]. Diciembre 2015, [30 de diciembre de 2015], https://www.sqlshack.com/how-to-install-sql-server-2014-management-studio/

[68] Solicitud de ayuda en blog “Infinite Scrolling in React using Apollo and React-Virtualized—GraphQL Cursor Pagination”, [en línea]. Agosto 2018, [18 de Agosto de 2018], https://medium.com/@andynedine/amazing-e9f4d6920e4e

[69] Solicitud de ayuda en Issues del proyecto en GitHub “GraphQLPagination”, [en línea]. Agosto 2018, [26 de Agosto de 2018], https://github.com/Gethyl/GraphQLPagination/issues/3

[70] Solicitud de ayuda en Issues del proyecto en GitHub “Infinite scroll using submit button”, [en línea]. Agosto 2018, [25 de Agosto de 2018],

https://github.com/dai-shi/react-apollo-github-api-infinite-scroll-example/issues/1

[71] Solicitud de ayuda en blog “React Apollo Client GraphQL Cursor Infinite Scroll Pagination with GitHub API”, [en línea]. Agosto 2018, [27 de Agosto de 2018], https://medium.com/@andynedine/awesome-easy-and-clean-thanks-a-lot-e27de398b85e

[72] Solicitud de ayuda en video-tutorial “Modelos, esquemas y resolvers en Mongoose y GraphQL”, [en línea]. Octubre de 2017, [julio de 2018], https://www.youtube.com/watch?v=ZHoBRxQYzpo

[73] “Registro de usuarios con GraphQL y React”, [en línea]. Diciembre 2017, [agosto de 2018], https://www.youtube.com/watch?v=x5ig2rvzi94

[74] Incidencia creada en StackOverflow – “Order root by name, and their images by position with Mongoose”, [en línea]. Agosto 2018, Agosto 2018, [12 de agosto de 2018], https://stackoverflow.com/questions/51802498/order-root-by-name-and-their-images-by-position-with-mongoose

[75] Incidencia creada en StackOverflow – “Ordenar en Mongoose artículos por nombre, y sus imágenes por <posición>”, [en línea]. Agosto 2018, Agosto 2018, [11 de agosto de 2018], https://es.stackoverflow.com/questions/188099/ordenar-en-mongoose-art%C3%ADculos-por-nombre-y-sus-im%C3%A1genes-por-posici%C3%B3n

[76] Incidencia creada en StackOverflow – “Listar productos con filtro de búsqueda en GraphQL y React”, [en línea]. Agosto 2018, Agosto 2018, [10 de agosto de 2018], https://es.stackoverflow.com/questions/187755/listar-productos-con-filtro-de-b%C3%BAsqueda-en-graphql-y-react

[77] Incidencia creada en StackOverflow – “Get values with GraphQL from a relation”, [en línea]. Julio 2017, [31 de julio de 2017], https://stackoverflow.com/questions/51619074/get-values-with-graphql-from-a-relation

[78] Incidencia creada en StackOverflow – “Update GraphQL variables after submit”, [en línea]. Agosto 2018, [8 de agosto de 2018], https://stackoverflow.com/questions/51754052/update-graphql-variables-after-submit

[79] “React js y el ciclo de vida de los componentes”, [en línea]. Agosto 2015, [14 de agosto de 2015], https://medium.com/@pedroparra/react-js-y-el-ciclo-de-vida-de-los-componentes-5d083e5089c6

[80] “Curso NodeJS y MongoDB. Crear un API REST JSON desde cero”, [en línea].Septiembre 2016, septiembre 2018, [16 de septiembre de 2018], https://www.youtube.com/playlist?list=PLUdlARNXMVkk7E88zOrphPyGdS50Tadlr

[81] “Clon de Instagram con NodeJS, GraphQL, React, MongoDB”, [en línea]. Octubre 2017, [20 de octubre de 2017], https://github.com/danielniquet/instagram-clone

[82] “The Heroku CLI”, [en línea], https://devcenter.heroku.com/articles/heroku-cli

[83] “Create-react-app with a Node server on Heroku”, [en línea]. 2016, [14 de Agosto de 2018], https://github.com/mars/heroku-cra-node

[84] “Git y Visual Studio Code”, [en línea]. Diciembre 2017, [12 de diciembre de 2017], http://lemoncode.net/lemoncode-blog/2017/12/12/git-y-visual-studio-code

[85] “Qué es el stack MEAN y cómo escoger el major para ti”, [en línea]. Enero 2015, [19 de enero de 2015], https://www.campusmvp.es/recursos/post/Que-es-el-stack-MEAN-y-como-escoger-el-mejor-para-ti.aspx